VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

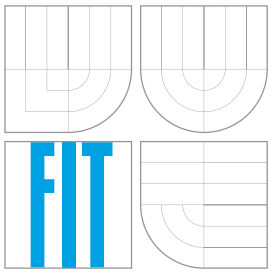
IMPLEMENTACE STATISTICKÝCH KOMPRESNÍCH METOD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ BENĚK
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IMPLEMENTACE STATISTICKÝCH KOMPRESNÍCH METOD

IMPLEMENTATION OF STATISTICAL COMPRESSION METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ BENĚK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DAVID BAŘINA

BRNO 2012

Abstrakt

Tato práce se zabývá statistickými kompresními metodami. V práci je popsán základní teoretický úvod do komprese dat a jsou zde popsáni nejdůležitější zástupci statistických kompresních metod. Také je zde popsán kompresní postup založený na Burrowsově-Wheelerově algoritmu, který je implementován formou přenositelné knihovny v jazyce C++. Práce srovnává účinnost jednotlivých metod na vhodném korpusu dat, který obsahuje různé typy souborů a podle dosažených výsledků diskutuje vhodnost použití metod a možná zlepšení.

Abstract

This thesis deals with the statistical compression methods. This work describes basic theoretical introduction to the data compression and the most important representatives of statistical methods. There is also described compression process based on the Burrows-Wheeler algorithm, which is implemented in the form of portable library in C++ programming language. This work compares effectiveness of each method on suitable data corpus, which contains various data types and according to achieved results discusses suitability of usage of methods and possible improvements.

Klíčová slova

Komprese dat, bezeztrátová komprese, statistické metody, BWT, MTF, Timestamp, IFC, WFC, RLE, Huffmanovo kódování, aritmetické kódování

Keywords

Data compression, lossless compression, statistical methods, BWT, MTF, Timestamp, IFC, WFC, RLE, Huffman coding, arithmetic coding

Citace

Tomáš Beněk: Implementace statistických kompresních metod, bakalářská práce, Brno, FIT VUT v Brně, 2012

Implementace statistických kompresních metod

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Davida Bařiny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Beněk
15. května 2012

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Davidovi Bařinovi za trpělivost a pomoc při řešení této práce.

Obsah

1 Úvod **2**

2 Teorie komprese dat **3**
2.1 Typy kompresních metod 3
2.2 Kompresní výkonnost 4
2.3 Historie komprese 4
2.4 Statistické metody komprese dat 5
2.5 Huffmanovo kódování 5
2.6 Aritmetické kódování 7
2.7 Burrowsův-Wheelerův kompresní algoritmus 10
2.8 Burrowsova-Wheelerova transformace 10
2.9 Metody GST 12
2.10 Kódování délek sledů 16
2.11 Entropické kódování 17

3 Implementace a testování **18**
3.1 Implementace 18
3.2 Testování 18
3.3 Testovací postup 20
3.4 Výsledky testování 20

4 Závěr **28**

A Obsah CD **30**

B Výsledky testů **31**

Kapitola 1

Úvod

S rozšiřováním a zpřístupňováním výpočetní techniky a narůstajícím počtem jejich uživatelů narůstá také datový objem, který je nutné ukládat nebo přenášet. Ovšem kapacita médií, na která se tato data ukládají, a propustnost komunikačních linek je omezená. Proto nastala otázka, zda je možné objem těchto dat nějakým způsobem redukovat. Tím se dostala ke slovu komprese dat, která umožňuje dosažení menší velikosti ukládaných a přenášených dat, čímž se jí dostalo velké důležitosti v mnoha odvětvích, které pracují s pomocí výpočetní techniky, například v telekomunikaci nebo při zálohování dat.

Tato práce se zabývá právě problematikou komprese dat, konkrétně statistickými kompresními metodami. Uvedení do problematiky je provedeno ve druhé kapitole. Jsou zde vysvětleny základní pojmy komprese dat a popsány základní typy kompresích metod. Ve třetí kapitole jsou blíže vysvětleny statistické kompresní metody. Ve čtvrté kapitole je popsán Burrowsův-Wheelerův kompresní algoritmus a postup při kompresi dat založený na této metodě. Pátá kapitola obsahuje informace k implementaci jednotlivých metod v této práci. V šesté kapitole jsou popsány metody testování kompresních metod a také postup a výsledky testování metod použitých v této práci. Sedmá kapitola obsahuje zhodnocení dosažených výsledků a možnosti pro zlepšení.

Kapitola 2

Teorie komprese dat

Definice pojmů v této kapitole jsou převzaty z [11].

Komprese dat je proces překódování vstupního (zdrojového) toku dat na jiný datový tok (výstupní, komprimovaný), který má menší velikost. Převod mezi těmito toky se nazývá kódování, což je proces vzájemně jednoznačného přiřazení symbolů jedné množiny symbolům druhé množiny. Komprese dat se dosáhne redukcí nebo odstraněním redundance v datech. Při kódování se buď redundance kódu nemění, může se zvyšovat (například při překódování do kódu pro opravu chyb) nebo se snižuje – dochází ke kompresi. Důvody pro kompresi mohou být např. limitovaná kapacita paměti nebo šířka přenosového pásma.

2.1 Typy kompresních metod

Kompresní metody se mohou dělit na ztrátové a bezeztrátové metody. U ztrátových metod se v průběhu komprese ztratí určité informace, což vede k lepšímu kompresnímu poměru, ale za cenu toho, že dekompresí už není možno získat zpět původní datový tok. Ztrátové komprese se využívají hlavně ke kompresi zvuku, obrazu a videa, kde nedokonalé lidské vnímání do určité míry ztrátu informace nepozná. Naopak bezeztrátové komprese umožňují dekompresí získat původní datový tok, což je nutné u určitých typů souborů, například u textových a zdrojových souborů, avšak dosahují horšího kompresního poměru.

Základními skupinami metod bezeztrátové komprese jsou statistické a slovníkové metody. Slovníkové metody využívají speciální datovou strukturu, slovník, do které se ukládají fragmenty komprimovaných dat. Pokud se při kompresi najde fragment, který je shodný s jednou položkou ve slovníku, do komprimovaného toku se místo fragmentu zapíše ukazatel na tuto položku. Mezi významné slovníkové metody patří např. LZ77, LZ78, LZW nebo LZMA. Statistickým metodám, mezi které patří i kontextové metody využívající predikce následujících symbolů, je věnována tato práce, a proto budou podrobněji popsány později.

Dalším rozdělením může být na metody neadaptivní, semiadaptivní a adaptivní. Neadaptivní metoda je fixní a v průběhu komprese nemění postup. Pracuje s určitým předpočítaným modelem, který je vhodný pouze pro určitý typ dat, a proto je její použití úzce specializované, protože pro jiný typ dat by podávala špatné výsledky. Semiadaptovní metoda v průběhu komprese také nemění postup, model je ale získán na zpracovávaných datech. Tyto metody většinou bývají dvouprůchodové, kdy při prvním průchodu jsou získány informace o vstupních datech a při druhém průchodu je provedena samotná komprese. Tento postup je však časově náročný a někdy ani není možný. Adaptivní metoda zkoumá vstupní data a na základě jejich analýzy v průběhu komprese mění postup.

2.2 Kompresní výkonnost

K určení kompresní výkonnosti se využívá několik veličin. Jednou z nich je kompresní poměr:

$$kompresní\ poměr = \frac{velikost\ výstupního\ toku}{velikost\ vstupního\ toku} \tag{2.1}$$

Hodnota v rozmezí 0–1 znamená kompresi, například při poměru 0,6 data po kompresi zabírají pouze 60 % původního objemu. Hodnota vyšší než 1 znamená negativní kompresi, tj. že data zabírají po kompresi více objemu než před kompresí.

Převrácenou hodnotou kompresního poměru získáme kompresní faktor:

$$kompresní\ faktor = \frac{velikost\ vstupního\ toku}{velikost\ výstupního\ toku} \tag{2.2}$$

Zde hodnoty menší než 1 znamenají expanzi a hodnoty vyšší znamenají kompresi, tedy čím vyšší kompresní faktor, tím lepší komprese.

Z kompresního poměru lze také získat hodnotu bpc (bits per character), tedy počet výstupních bitů potřebných k zakódování jednoho znaku.

$$bpc = \frac{velikost\ výstupního\ toku \times 8}{velikost\ vstupního\ toku} \tag{2.3}$$

Kompresní zisk:

$$kompresní\ zisk = 100 \cdot \log_e \cdot \frac{referenční\ velikost}{komprimovaná\ velikost} \tag{2.4}$$

Referenční velikost může být buď velikost vstupního toku, nebo velikost komprimovaného toku vytvořeného nějakou standardní kompresní metodou. Slouží k porovnávání metod, protože díky přítomnosti logaritmu se mohou dva kompresní zisky porovnat pomocí rozdílu hodnot. Jednotkou kompresního zisku je procentuální logaritmický poměr [o/o].

Pro měření rychlosti komprese byla zavedena veličina počet cyklů na byte (cpb). Je to střední počet strojových cyklů potřebných ke kompresi jednoho bytu.

2.3 Historie komprese

Problematika komprese dat je už poměrně stará, vznikla daleko dříve před vznikem počítačů. Nejstaršími metodami jsou Braillův a Morseův kód. Hlavní rozvoj kompresních metod ale nastal až v posledních patnácti letech, protože i levné osobní počítače a jiná výpočetní technika dosáhly dostatečné výkonnosti k provádění náročných komprimačních metod.

Braillův kód vytvořil v roce 1820 L. Braille a je dodnes používán, protože umožňuje číst zrakově postiženým lidem (písmo je pro ně vytvářeno pomocí výstupků, které mohou rozpoznat hmatem). Braillův kód je tvořen skupinami 3×2 bodů. Každý bod nese informaci jednoho bitu, což dává celkem 6 bitů na jeden obrazec, tedy 64 různých obrazců. Objem komprese pomocí tohoto kódu je malý, přesto je významný, například při úspoře místa v knihách pro zrakově postižené.

Morseův kód vymyslel S. Morse v roce 1843. Kód je používán při telegrafních přenosech, nejrozšířenější je na lodích. Je zde využito kódu s proměnlivou délkou, kdy znakům, které se vyskytují častěji je přiřazen kratší kód a naopak. Zakódování je prováděno pomocí teček a čárek.

2.4 Statistické metody komprese dat

Statistické metody jsou postaveny na frekvenci výskytu jednotlivých symbolů a na kódech s proměnnou délkou. Symbolům s vyšší frekvencí (pravděpodobností) výskytu přidělují kód kratší délky a symbolům s nižší frekvencí výskytu delší kód.

Statistické metody jsou složeny ze dvou stupňů: modelovacího a kódovacího. Model přiděluje vstupním symbolům pravděpodobnosti a kódovací stupeň pak skutečně kóduje symboly na základě jejich pravděpodobností. Model může být statický, nebo dynamický (adaptivní). Statický model se v průběhu komprese nemění, zatímco adaptivní model se v průběhu komprese neustále aktualizuje.

Hlavními dvěma kódovacími metodami jsou Huffmanovo kódování a aritmetické kódování.

2.5 Huffmanovo kódování

Huffmanovo kódování [11] je jednou z nejstarších metod komprese dat. Vytvořil ji v roce 1952 D. Huffman. Tato metoda se stala všeobecně používanou při bezeztrátové kompresi dat, často na konci řetězce různých metod. Využívá prefixového kódu. Nejlepší výsledky dává pro pravděpodobnosti symbolů rovné záporným mocninám čísla 2.

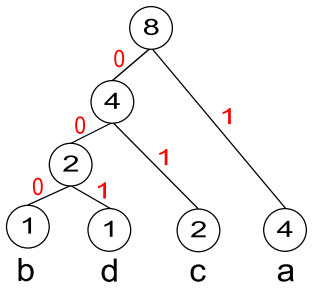
Metoda při kódování vytváří seznam symbolů v sestupném pořadí jejich pravděpodobností. Poté je zkonstruován strom se symbolem v každém listu zdola nahoru. To je prováděno v krocích. V každém kroku se vyberou dva symboly s nejnižší pravděpodobností, sečtou se do vrcholu místního sbíhání větví, vyškrtnou se ze seznamu a nahradí se symbolem, který reprezentuje tento součet. Když je strom redukován na jediný přidaný symbol reprezentující celou abecedu, je konstrukce stromu hotová. Jednotlivým hranám je poté přiděleno hodnocení a strom se poté systematicky prochází a zjišťují se tak kódy pro jednotlivé symboly.

Jako příklad uvedeme řetězec „abcaacda“, viz. tabulka 2.1.

Symbol	Frekvence	Pravděpodobnost
a	4	0,5
c	2	0,25
b	1	0,125
d	1	0,125

Tabulka 2.1: Pravděpodobnostní model dat řetězce „abcaacda“.

Nyní zkonstruujeme Huffmanův strom, viz. obrázek 2.1.



Obrázek 2.1: Huffmanův strom pro řetězec „abcaacda“

Jednotlivým symbolům je přidělen kód, viz. tabulka 2.2.

Symbol	Kód
a	1
c	01
b	000
d	001

Tabulka 2.2: Kódy přiřazené jednotlivým symbolům po proběhnutí kódování.

Vstupní řetězec „abcaacda“ bude tedy zakódován jako 10000111010011.

Při dekódování je nutné mít k dispozici buď pravděpodobnosti jednotlivých symbolů nebo stejný Huffmanův strom, který byl využit při kódování. Poté už je dekódování jednoduché. Začíná se v kořenu stromu a přečte se bit ze vstupního toku. Podle hodnoty tohoto bitu dekodér prochází stromem a čte další bity, a podle jejich hodnot opět postupuje stromem dokud se nedostane k listu. Dospěním k listu stromu je určen nekomprimovaný původní kód symbolu, a ten je odeslán na výstup. Poté se znovu pro nový bit začíná v kořenu stromu.

2.5.1 Adaptivní Huffmanovo kódování

Huffmanova metoda předpokládá, že jsou známé všechny četnosti výskytů symbolů. Ty lze získat průchodem celého souboru před vlastní kompresí. Tato metoda je semiadaptivní, je však pro praxi příliš pomalá. Také je možné stanovit četnosti napevno podle určitých kritérií, tento způsob však nepovede k optimálnímu kódu pro všechny typy souborů. Proto se využívá adaptivního Huffmanova kódování.

Hlavní změnou je, že se začíná s prázdným Huffmanovým stromem a upravuje se v průběhu čtení a zpracování symbolů. První symbol se zapíše v nekomprimované formě do výstupního toku. Také se tento symbol přidá do stromu a je mu přidělen kód. Pokud je tento symbol opět přečten, do výstupního toku se zapíše jeho kód a frekvence tohoto symbolu se inkrementuje. Tímto se strom modifikuje a je nutné ověřit, zda se stále jedná o Huffmanův strom. Pokud ne, je nutné provést jeho přeuspořádání, čímž se změní také kódy symbolů.

Kritériem toho, zda se jedná o Huffmanův strom, je splnění vlastnosti sourozenců (sibling property). Tato vlastnost znamená, že pokud procházíme stromem po úrovních a procházíme je zleva doprava a zdola (listy) nahoru (kořen), frekvence jsou uspořádány v neklesajícím pořadí, tedy levý dolní uzel má nejnižší frekvenci a pravý horní uzel má nejvyšší frekvenci. Toto je důležité, protože symbol s vysokou frekvencí musí mít přidělen kratší kód a proto musí být ve stromu umístěn vysoko. Požadavek seřazení frekvencí na všech úrovních zleva doprava je umělý, zjednodušuje však proces aktualizace stromu.

Aktualizace stromu probíhá následovně. Cyklus začíná v aktuálním uzlu (odpovídá právě přečtenému symbolu). Tento uzel je list X s frekvencí výskytu F. Každá iterace sestává ze tří kroků:

1. Porovnej X s jeho následovníky ve stromu (zleva doprava a zdola nahoru). Pokud má bezprostřední následovník frekvenci $F+1$ nebo vyšší, uzly jsou uspořádané a nemusí se nic měnit. V opačném případě mají někteří následovníci stejné nebo menší frekvence jako X. V takovém případě se X musí prohodit musí prohodit s posledním uzlem v této skupině (nesmí se však prohodit se svým rodičem).

2. Inkrementuj frekvenci výskytů uzlu X a frekvence všech jeho rodičů o 1.

3. Pokud je X kořenem, cyklus končí, jinak opakuj cyklus s rodiči uzlu X.

Dekompresor zrcadlí stejné kroky. Při přečtení nekomprimované formy symbolu jej zapíše do výstupního toku, přidá do stromu a přidělí mu kód. Při přečtení symbolu v komprimované formě použije aktuální strom a přečtený kód k určení, o jaký symbol se jedná, tento symbol zapíše do výstupního toku a upraví strom stejně jako kompresor. Aby dekodér poznal, v jaké formě je přečtený symbol, tak se před každý nekomprimovaný symbol vloží speciální kód změny (escape code) s proměnnou délkou. Když dekompresor přečte tento kód, ví, že bude následovat symbol v nekomprimované podobě.

Při použití adaptivního Huffmanova kódování mohou nastat dva problémy – přeplnění čítače [11] a přeplnění kódu [11]. Problém přeplnění čítače je takový, že hodnoty frekvencí ve stromu se ukládají v polích s pevnou délkou. Tato pole se však mohou přeplnit. Řešením je změnit měřítko všech čítačů celočíselným vydělením 2 vždy, když čítač kořene dosáhne maximální hodnoty. To se provede tak, že se podělí pouze čítače listů a následnou aktualizací čítačů vnitřních uzlů. Problémem tohoto řešení je snížení přesnosti kvůli celočíselnému dělení, což může vést ke změně Huffmanova stromu na strom nesplňující vlastnost sourozenců. Problém přeplnění kódu je vážnější než přeplnění čítače. Může nastat, pokud se do stromu přidá mnoho symbolů a strom je poté vysoký. Řešením může být ukládání bitů kódu do vázaného seznamu. Toto řešení je poté omezeno pouze velikostí poskytnuté paměti, je však pomalé. Dalším řešením je akumulace kódů ve velké celočíselné proměnné (např. 64 bitů široké) a maximální velikost kódu (64 bitů) uvést jako omezení programu.

2.6 Aritmetické kódování

Huffmanova metoda je jednoduchá a účinná, avšak jediným případem, kdy vytvoří ideální kódy s proměnnou délkou je tehdy, pokud mají symboly pravděpodobnosti výskytu rovny záporným mocninám 2. Je to proto, že Huffmanova metoda přiděluje symbolům abecedy kódy s celočíselným počtem bitů. Aritmetické kódování [11] překonává tento problém tím, že přidělí jeden kód celému vstupnímu souboru. Metoda začíná s jistým intervalem, vstupní tok čte symbol po symbolu a na základě pravděpodobností výskytu jednotlivých symbolů provádí zužování tohoto intervalu. Čím menší je tento interval, tím více bitů je potřeba k jeho reprezentaci. Aby se dosáhlo komprese symboly s vyšší pravděpodobností zúží interval méně než symboly s nižší pravděpodobností. Díky tomu symboly s vyšší pravděpodobností do výstupu přidávají méně bitů. Interval lze vyjádřit buď dolní a horní mezí, nebo jednou mezí a šířkou intervalu.

Hlavní kroky aritmetického kódování jsou následující:

1. Definuj aktuální interval $I = \langle 0; 1)$.
2. Pro každý symbol s vstupního toku opakuj:
   (a) Rozděl aktuální interval na podintervaly, jejichž velikosti odpovídají pravděpodobnostem symbolů.
   (b) Vyber podinterval pro s a prohlas jej za nový aktuální interval.
3. Po zpracování celého vstupního toku bude výstupem číslo, které jednoznačně identifikuje aktuální interval (tj. jakékoliv číslo uvnitř intervalu).

Pro znázornění principu kódování bude uvedeno kódování na řetězci *abcaacda*, viz. tabulka 2.3.

Symbol	Frekvence	Pravděpodobnost	Interval	Kumulovaná frekvence
a	4	0,5	$\langle 0; 0{,}5)$	0
c	2	0,25	$\langle 0{,}5; 0{,}75)$	4
b	1	0,125	$\langle 0{,}75; 0{,}875)$	6
d	1	0,125	$\langle 0{,}875; 1)$	7
Celková kumulovaná frekvence				8

Tabulka 2.3: Pravděpodobnostní model dat řetězce „abcaacda“.

Takto vytvořený pravděpodobnostní model je zapsán do výstupního toku před komprimovaným kódem a je prvním vstupem dekodéru. Kódovací proces začíná definicí dvou proměnných *Low* a *High* určujících interval a jejich nastavením na 0 a 1. Poté se hodnoty těchto proměnných upravují podle následujících vzorců:

$$High = OldLow + (OldHigh - OldLow) \times HighRange(X) \quad (2.5)$$

$$Low = OldLow + (OldHigh - OldLow) \times LowRange(X), \quad (2.6)$$

Zde $HighRange(X)$ a $LowRange(X)$ označují horní a dolní mez symbolu X. Tyto proměnné a úprava jejich hodnot při kódování řetězce *abcaacda* jsou znázorněny v tabulce 2.4. Výstupem kódování je číslo *Low*, tedy 0,4090576172 (binárně 0,01101000101), ale do výstupního kódu se zapíše pouze desetinná část, tedy 4090576172 (binárně 01101000101).

Symbol	Proměnná	Výpočet nových hodnot
a	High	$0 + (1 - 0) \times 0{,}5 = 0{,}5$
	Low	$0 + (1 - 0) \times 0 = 0$
b	High	$0 + (0{,}5 - 0) \times 0{,}875 = 0{,}4375$
	Low	$0 + (0{,}5 - 0) \times 0{,}75 = 0{,}375$
c	High	$0{,}375 + (0{,}4375 - 0{,}375) \times 0{,}75 = 0{,}421875$
	Low	$0{,}375 + (0{,}4375 - 0{,}375) \times 0{,}5 = 0{,}40625$
a	High	$0{,}40625 + (0{,}421875 - 0{,}40625) \times 0{,}5 = 0{,}4140625$
	Low	$0{,}40625 + (0{,}421875 - 0{,}40625) \times 0 = 0{,}40625$
a	High	$0{,}40625 + (0{,}4140625 - 0{,}40625) \times 0{,}5 = 0{,}41015625$
	Low	$0{,}40625 + (0{,}4140625 - 0{,}40625) \times 0 = 0{,}40625$
c	High	$0{,}40625 + (0{,}41015625 - 0{,}40625) \times 0{,}75 = 0{,}4091796875$
	Low	$0{,}40625 + (0{,}41015625 - 0{,}40625) \times 0{,}5 = 0{,}408203125$
d	High	$0{,}408203125 + (0{,}4091796875 - 0{,}408203125) \times 1 = 0{,}4091796875$
	Low	$0{,}408203125 + (0{,}4091796875 - 0{,}408203125) \times 0{,}875 = 0{,}4090576172$
a	High	$0{,}4090576172 + (0{,}4091796875 - 0{,}4090576172) \times 0{,}5 = 0{,}4091186524$
	Low	$0{,}4090576172 + (0{,}4091796875 - 0{,}4090576172) \times 0 = 0{,}4090576172$

Tabulka 2.4: Proces aritmetického kódování řetězce „abcaacda“.

Proces dekódování probíhá opačně. Dekodér nejprve načte pravděpodobnostní model, který kodér vložil před komprimovaná data. První načtenou číslicí je 4, takže dekodér ví, že celý kód je číslo začínající 0,4... Toto číslo je v podintervalu $\langle 0; 0{,}5)$, které odpovídá symbolu a. K vyloučení vlivu symbolu a dekodér použije následující operace:

$$Code = \frac{Code - LowRange(X)}{Range} \quad (2.7)$$

Odečtením dolní meze 0,5 symbolu a a vydělením šířkou podintervalu $a(0,5)$ získáme číslo 0,8181152344, které leží v podintervalu $\langle 0{,}75; 0{,}875)$, takže dekodér ví, že dalším symbolem je symbol b.

Průběh dekódování je ukázán v tabulce 2.5.

Symbol	Podinterval	Kód - dolní mez	Rozsah	Výsledek
a	$\langle 0; 0{,}5)$	$0{,}4090576172 - 0 = 0{,}4090576172$	0,5	0,8181152344
b	$\langle 0{,}75; 0{,}875)$	$0{,}8181152344 - 0{,}75 = 0{,}0681152344$	0,125	0,5449218752
c	$\langle 0,5; 0{,}75)$	$0{,}5449218752 - 0{,}5 = 0{,}0449218752$	0,25	0,1796875008
a	$\langle 0; 0{,}5)$	$0{,}1796875008 - 0 = 0{,}1796875008$	0,5	0,3593750016
a	$\langle 0; 0{,}5)$	$0{,}3593750016 - 0 = 0{,}3593750016$	0,5	0,7187500032
c	$\langle 0{,}5; 0{,}75)$	$0{,}7187500032 - 0{,}5 = 0{,}2187500032$	0,25	0,8750000128
d	$\langle 0{,}875; 1)$	$0{,}8750000128 - 0{,}875 = 0{,}0000000128$	0,125	0,0000001024
a	$\langle 0; 0{,}5)$	$0{,}0000001024 - 0 = 0{,}0000001024$	0,5	0,0000002048

Tabulka 2.5: Proces aritmetického dekódování řetězce „abcaacda“.

Aby dekodér poznal, kdy má skončit, lze použít speciálního symbolu konce souboru nebo předat informaci o délce zakódované zprávy.

2.6.1 Implementace

Protože výše popsaná metoda předpokládá možnost využití čísel s neomezenou přesností, nelze ji využít v praxi, protože dnešní počítače mají omezený rozsah datových typů a výpočty v plovoucí desetinné čárce jsou náročné. Proces dekódování je také nepraktický, protože kód, který je reprezentován jedním číslem, může být velmi dlouhý a dělení takového čísla je složité.

Proto se při praktické implementaci [11] používají pouze celá čísla, která nejsou příliš dlouhá (kvůli přesnosti). Využívá se opět dvou proměnných Low a $High$, které udržují dolní a horní mez aktuálního podintervalu. Jakmile se nejlevější číslice těchto proměnných shodnou, jsou odsunuty ven z těchto proměnných a tato číslice je zapsána do výstupního toku. Díky tomu hodnoty těchto proměnných nemohou narůst do příliš velkých hodnot. Když dojde k vysunutí shodné číslice, do proměnné Low se zprava doplní číslo 0 a do proměnné $High$ se zprava doplní číslo 9. Problémem je, že proměnná $High$ se musí inicializovat na 1, ale obsah proměnných se musí interpretovat jako zlomky v intervalu $\langle 0; 1)$. Řešením je inicializovat tuto proměnnou na 9999.

Dekódování je opakem kódování. Proměnné se inicializují takto: $Low = 0000$, $High = 9999$ a $Code = (\textit{první číslice komprimované zprávy})$. V každém cyklu se provedou následující kroky:

1. Vypočítej $Index = \frac{(Code - Low + 1 \cdot TotalCumFreq - 1)}{High - Low}$ a odsekni jej k nejbližšímu celému číslu.

2. Použij $Index$ k nalezení následujícího symbolu pomocí kumulovaných pravděpodobností v tabulce pravděpodobnostního modelu.

3. Aktualizuj proměnné Low a $High$ pomocí vzorců ($LowCumFreq(X)$ a $HighCumFreq(X)$ jsou kumulované frekvence symbolu X a symbolu nad ním, $TotalCumFreq$ je celková

kumulovaná frekvence):

$$High = Low + (High - Low + 1) \cdot \frac{HighCumFreq(X)}{TotalCumFreq} \quad (2.8)$$

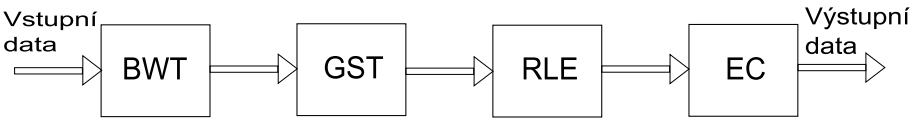
$$Low = Low + (High - Low + 1) \cdot \frac{LowCumFreq(X)}{TotalCumFreq} \quad (2.9)$$

4. Pokud jsou nejlevější číslice Low a $High$ shodné, posuň proměnné Low, $High$ a $Code$ o jednu pozici doleva. Do Low zprava přidej 0, do $High$ 9 a do $Code$ následující vstupní číslici z komprimované zprávy.

Protože počet symbolů může být velký, vznikl by problém s jeho uložením. Proto se jako indikátor konce souboru využívá speciálního symbolu přidaného na konec zakódované zprávy.

2.7 Burrowsův-Wheelerův kompresní algoritmus

Burrowsův-Wheelerův kompresní algoritmus je založen na Burrowsově-Wheelerově transformaci. Po transformaci jsou dále provedeny algoritmy, které přispívají ke zlepšení kompresního poměru. Tyto algoritmu jsou prováděny po sobě, čímž tvoří kompresní řetězec. Kroky tohoto řetězce jsou zobrazeny na obrázku 2.2.



Obrázek 2.2: Schéma kompresního algoritmu Burrows-Wheeler. Jednotlivé fáze schématu: BWT – Burrowsova-Wheelerova transformace, GST – Global structure transformation, RLE – Run lenght encoding (Kódování délek sledů), EC – Entropické kódování. Ve fázi Burrowsovy-Wheelerovy transformace je načten blok vstupních dat a provede se na něm Burrowsova-Wheelerova transformace. Počet symbolů po této fázi zůstává stejný, ovšem přibude další číslo – index, který je nutný ke zpětné transformaci. Fáze global structure transformation mění lokální kontext symbolů na globální kontext. Tato fáze opět zachovává počet symbolů stejný jako u vstupního bloku. Fáze kódování délek sledů pomocí metody kódování délek sledů snižuje počet symbolů. Fáze entropického kódování komprimuje symboly využitím adaptovaného modelu.

2.8 Burrowsova-Wheelerova transformace

Tato metoda slouží k transformaci dat [11]. Metoda zlepšuje kompresi transformací vstupních dat takovým způsobem, že vytváří shluky stejných symbolů, což je pro kompresi výhodné (například při použití metody kódování délek sledů). Metoda samotná data nekomprimuje. Metoda Burrows-Wheeler, na rozdíl od většiny ostatních metod pracujících v proudovém režimu, pracuje v blokovém režimu. Vstupní data jsou načítána blok po bloku a každý blok se zpracovává odděleně, jako jeden řetězec. Čím větší je velikost bloku, tím je větší šance na vytvoření shluků shodných symbolů a následně lepší kompresi. Tato metoda je obecně použitelná, dobře funguje pro text, obraz i zvuk. Poskytuje také velký potenciál ke zlepšení, protože je relativně nová a existují místa ke zlepšení řetězců metod na ní založených.

Transformace probíhá v následujících krocích:

1. Ze vstupu se načte řetězec S („ratata“) o velikosti n, a vytvoří se všechny jeho možné rotace v matici $n \times n$, viz. tabulka 2.6.

Index	F					L
0	r	a	t	a	t	a
1	a	r	a	t	a	t
2	t	a	r	a	t	a
3	a	t	a	r	a	t
4	t	a	t	a	r	a
5	a	t	a	t	a	r

Tabulka 2.6: Matice $n \times n$ rotací vstupního řetězce, kde n je velikost vstupního řetězce, F první sloupec matice, L poslední sloupec matice. Jednotlivé rotace vznikají postupnou rotací vstupního řetězce.

2. Tyto rotace se poté lexikograficky seřadí, viz. tabulka 2.7.

Index	F					L
0	a	r	a	t	a	t
1	a	t	a	r	a	t
2	a	t	a	t	a	r
3=I	r	a	t	a	t	a
4	t	a	r	a	t	a
5	t	a	t	a	r	a

Tabulka 2.7: Lexikograficky seřazená matice rotací, kde F první sloupec matice, L poslední sloupec matice a I je index původního řetězce v matici.

3. Výstupem transformace bude poslední sloupec L této seřazené matice a index I původního řetězce v matici, nutný ke zpětné transformaci.

Můžeme si všimnout, že poslední sloupec, který je výstupem metody, obsahuje shluky shodných symbolů, čímž pomáhá zlepšit kompresi. Také je pomocí něj a indexu původního řetězce v seřazené matici sestavit původní řetězec, což pomocí jiného sloupce není možné. Zpětná transformace probíhá v následujících krocích:

1. První sloupec F matice získáme lexikografickým seřazením posledního sloupce L, viz. tabulka 2.8.
2. Začíná se na řádku I
3. Na výstup se zašle symbol ze sloupce F na aktuálním řádku
4. Přesuň se na řádek, který má stejnou pozici ve sloupci L uvnitř množiny posledního symbolu než pozice posledního symbolu uvnitř množiny posledního symbolu ve sloupci F, např. pokud poslední symbol byl C a C byl pátým C ve sloupci F, přesuň se na řádek s pátým C ve sloupci L.

Index	L	F
0	t	a
1	t	a
2	r	a
3=I	a	r
4	a	t
5	a	t

Tabulka 2.8: Získání prvního sloupce matice F lexikografickým seřazením posledního sloupce L.

5. Opakuj 3. a 4. dokud nedosáhneme řádku I.

Protože metoda je účinná při velkých blocích n a matice $n \times n$ by byly obrovské, v praxi se ve skutečnosti v paměti nevytváří celá tato matice. Využívá se skutečnosti, že načtený řetězec v sobě obsahuje také všechny rotace. Proto stačí vytvořit pouze pomocné pole, které bude obsahovat ukazatele do původního pole a každý ukazatel bude ukazovat na začátek jedné rotace.

2.9 Metody GST

Fáze global structure transformation slouží ke změně lokálního kontextu symbolů ve vstupním řetězci na globální kontext celého řetězce.

2.9.1 Přesun na začátek

Tato kapitola byla převzata z [11]. Základní myšlenkou tohoto postupu je udržovat abecedu A symbolů jako seznam, ve kterém jsou často se vyskytující symboly umístěny na začátku. Každý prvek seznamu je číslem vyjadřujícím počet symbolů, které mu předcházejí. Takže pokud $A = \{a, b, c, d\}$ a má-li se kódovat symbol vstupního toku c, bude zakódován jako 2, protože jej předcházejí dva symboly. Výstupem metody jsou tedy tato čísla.

Způsob, jakým jsou často se vyskytující symboly umísťovány na začátek seznamu je jednoduchý. Při načtení symbolu ze vstupního toku je tento symbol v seznamu přesunut na začátek. Jako příklad vezmeme abecedu $A = \{a, b, c, d\}$ a vstupní řetězec *abcaaada*. Výstupem bude 01220031. Názorně je tento způsob ukázán v tabulce 2.9.

Proces dekódování využívá stejný princip. Jsou čteny čísla určující pozici symbolu v abecedě, výstupem bude pozice tohoto čísla v seznamu a po každém načteném čísle je symbol přesunut na začátek seznamu. Jako příklad vezmeme opět abecedu $A = \{a, c, b, d\}$ a vstupním řetězcem bude 01220031. Jako výstup získáme původní řetězec *abcaaada*. Postup je ukázán v tabulce 2.10.

Metoda je lokálně adaptivní, protože se adaptuje podle počtu výskytů symbolů v lokálních oblastech zpracovávaných symbolů. Předpokládá výskyt více stejných symbolů za sebou, což se nazývá koncentrační vlastnost. Pouze za předpokladu, že vstupní data splňují koncentrační vlastnost dává metoda dobré výsledky, protože tak může symbolům dávat nižší hodnoty, což přispívá k lepší kompresi při použití Huffmanova nebo aritmetického kódování, protože nižším číslům lze přiřadit kratší kód. Výhodou metody je její jednoduchá implementace a časová nenáročnost.

Vstupní symbol	Abeceda	Výstupní symbol
a	A = {a, b, c, d}	0
b	A = {a, b, c, d}	1
c	A = {b, a, c, d}	2
a	A = {c, b, a, d}	2
a	A = {a, c, b, d}	0
a	A = {a, c, b, d}	0
d	A = {a, c, b, d}	3
a	A = {d, a, c, b}	1

Tabulka 2.9: Kódování řetězce *abcaaada* metodou přesun na začátek. V průběhu kódování vstupních symbolů se mění abeceda přesouváním zpracovávaných symbolů na její začátek.

Vstupní symbol	Abeceda	Výstupní symbol
0	A = {a, b, c, d}	a
1	A = {a, b, c, d}	b
2	A = {b, a, c, d}	c
2	A = {c, b, a, d}	a
0	A = {a, c, b, d}	a
0	A = {a, c, b, d}	a
3	A = {a, c, b, d}	d
1	A = {d, a, c, b}	a

Tabulka 2.10: Dekódování řetězce *abcaaada* metodou přesun na začátek. V průběhu dekódování vstupních symbolů se mění abeceda přesouváním zpracovávaných symbolů na její začátek.

Protože metoda přesuň na začátek přesouvá na začátek abecedy úplně každý symbol nezávisle na tom, jak častý byl jeho předcházející výskyt, vzniklo několik modifikací řešících tento problém. Modifikace MTF-1 je upravena tak, že na první pozici v abecedě je přesunut pouze znak nacházející se na druhé pozici, znaky na pozicích vyšších než druhé jsou přesunuty na druhou pozici. Modifikace MTF-2 je dále upravena tak, že znak z druhé pozice se přesouvá na první, pouze pokud předchozí znak nebyl na první pozici.

2.9.2 Kódování intervalu

Metoda kódování intervalu (interval encoding) [10] je založena na vzdálenostech výskytů symbolů. Pro každý symbol ze vstupního toku x_i^{bwt} se hledá jeho další výskyt ve vstupním toku x_p^{bwt}. Výstupem poté bude vzdálenost (nebo také počet symbolů mezi nimi) k tomuto dalšímu výskytu, tedy $p-i$. Pokud zde nejsou další výskyty tohoto symbolu, výstupem bude 0. Abychom mohli opět získat původní řetězec, musíme znát první pozice všech symbolů vstupní abecedy. Výhodou této metody je jednoduchá implementace a rychlost.

V tabulce 2.11 je vidět příklad výstupu metody na řetězci *abcaacdad*:

Vstupní symbol	První výskyt	Vzdálenost k dalšímu výskytu
a	1	3,1,3,0
b	2	0
c	3	3,0
d	7	2,0

Tabulka 2.11: Kódování metodou kódování intervalu pro řetězec *abcaacdad*. Pro každý symbol musíme znát jeho první výskyt, a poté se kódují vzdálenosti mezi výskyty symbolu. Pokud už ve vstupním řetězci nejsou další výskyty symbolu, výstupem je 0.

2.9.3 Kódování vzdálenosti

Metoda kódování vzdálenost (distance coding)i [6] je také založena na vzdálenostech výskytů symbolů, podobně jako metoda kódování intervalu. Pro každý symbol ze vstupního toku x_i^{bwt} se hledá jeho další výskyt ve vstupním toku x_p^{bwt}. Výstupem poté bude vzdálenost (nebo také počet symbolů mezi nimi) k tomuto dalšímu výskytu, tedy $p - i$. Pokud zde nejsou další výskyty tohoto symbolu, výstupem bude 0. Abychom mohli opět získat původní řetězec, musíme znát první pozice všech symbolů vstupní abecedy. Hlavní rozdíl oproti metodě kódování intervalu spočívá v tom, že se zde nepočítají známé symboly a přeskakují se opakující se symboly.

V tabulce 2.12 je vidět příklad výstupu metody na řetězci *abcaacdad*:

2.9.4 Inverzní frekvence

Metoda inverzní frekvence (inversion frequencies) [4] je také založena na vzdálenostech výskytů symbolů. Pro každý symbol a v abecedě je prohledáván blok vstupních dat X_{in}. Pokud jde o první výskyt tohoto znaku, výstupem je jeho pozice v X_{in}. Pro další výskyty platí, že pokud je současně zpracovávaný znak v X_{in} shodný se současně zkoumaným symbolem a, výstupem je počet symbolů větších než a mezi současnou a poslední pozicí výskytu symbolu a v X_{in}. Pří dekódování musíme navíc znát také počty výskytů jednotlivých znaků abecedy v bloku vstupních dat, čímž je zvyšována velikost výstupu.

Vstupní symbol	První výskyt	Vzdálenost k dalšímu výskytu
a	1	$3, 3, 0$
b	2	0
c	3	$3, 0$
d	7	$2, 0$

Tabulka 2.12: Kódování metodou kódování vzdálenosti pro řetězec *abcaacdad*. Pro každý symbol musíme znát jeho první výskyt, a poté se kódují vzdálenosti mezi výskyty symbolu, přičemž opakující se symboly se přeskakují. Pokud už ve vstupním řetězci nejsou další výskyty symbolu, výstupem je 0.

2.9.5 Seřazené inverzní frekvence

Metoda seřazené inverzní frekvence (SIF – sorted inversion frequencies) [2] vylepšuje metodu inverzní frekvence. V metodě inverzní frekvence je pro každý symbol ve vstupních datech produkován počet symbolů větších než tento symbol mezi jeho současnou a poslední pozicí výskytu. Protože jsou zde počítány pouze symboly větší, tak pokud jsou první zpracovávány symboly s vysokým počtem výskytů, budou poté počty následujících symbolů s nižším počtem výskytů nabývat nižších hodnot. Na druhou stranu počty symbolů s vysokým počtem výskytů jsou vyšší než pro symboly s nižším počtem výskytů. Na kompresi má tedy vliv permutace vstupní abecedy podle frekvence výskytů symbolů. Ta může být seřazena vzestupně nebo sestupně, pro každý typ souboru se však účinnost může lišit podle tohoto seřazení. Abychom nalezli optimální směr řazení, využívá se rozdělení symbolů. Nechť A_{in} je vstupní abeceda, X_{in} je vstupní tok dat, n je délka vstupního toku, a je symbol vstupní abecedy, f_a je počet výskytů v X_{in}, tj. rozdělení symbolu. Poté S popisuje podíl symbolu a v A_{in}, pro který platí, že počet jeho výskytů v X_{in} je dvakrát větší než průměrný počet výskytů v X_{in} délky n:

$$S = 100 \cdot \frac{|\{a|f_a > 2 \cdot \frac{n}{|A_{in}|}\}|}{|A_{in}|} \tag{2.10}$$

První se spočítá rozdělení všech symbolů abecedy spolu s odpovídající hodnotou S a poté je podle hodnoty S provedena permutace abecedy symbolů. Pokud je hodnota S větší nebo rovna 10, řadí se vzestupně, jinak se řadí sestupně.

2.9.6 Weighted frequency count

Metoda weighted frequency count [8] počítá výstupní hodnotu pro každý symbol pomocí funkce, která bere v úvahu počet výskytů symbolu a vzdálenost jeho posledního výskytu uvnitř posuvného okna. Každé pozici uvnitř posuvného okna je přidělena váha. Váhy bližších vzdáleností jsou vyšší než váhy větších vzdáleností. Pro každý symbol abecedy jsou váhy jeho výskytů uvnitř okna sečteny do odpovídajícího čítače. Seznam čítačů symbolů abecedy je seřazen sestupně, tzn. že nejvyšší čítač je na první pozici v seznamu. Tímto způsobem symboly, které se vyskytují častěji, získají nižší hodnotu než symboly vyskytující se méně často. Vážení a třídění musí být znovu provedeno pro každý zpracovávaný symbol, což vede k vyšší časové náročnosti metody.

2.9.7 Advanced weighted frequency count

Metoda advanced weighted frequency count [2] vznikla upravením váhové funkce metody weighted frequency count. Metoda weighted frequency count používá fixní váhy nezávisle na souboru. Protože se však struktura a rozdělení symbolů liší soubor od souboru, nepovede tento přístup k optimálním výsledkům pro všechny soubory. Pro některé soubory se funkce se silnými váhami nejvíce hodí pro symboly nedávné než pro starší symboly. Pro jiné soubory zase dává nejlepší výsledky váhová funkce, která přiděluje starším symbolům téměř stejné váhy jako symbolům nedávným. Proto metoda advanced weighted frequency count nepoužívá fixní váhy, ale počítá váhy v závislosti na rozdělení symbolů.

2.9.8 Incremental frequency count

Metoda incremental frequency count [1] řeší problém časové náročnosti metody weighted frequency count při snaze zachovat její dobré výsledky. Metoda také používá seznam čítačů symbolů a posuvné okno, změna je v jejich úpravách. Třídění zde není prováděno pro všechny čítače, ale pouze pro jeden čítač pro každý zpracovávaný symbol. Za účelem přidělit často se vyskytujícím symbolům a blízkým symbolům vyšší váhu než symbolům málo se vyskytujícím a vzdáleným může být čítač zvyšován nebo snižován, v průměru se zvyšuje, proto název metody Incremental frequency count.

2.9.9 Časové razítko

Metoda časové razítko (time stamp) [3] prochází blok vstupních dat, a pro každý symbol vypíše jeho pozici v abecedě. Abeceda je podobně jako u metody přesuň na začátek postupně upravována, a to tak, že se zpracovaný symbol a přesouvá před první prvek abecedy, který byl zpracováván nejvýše jednou od posledního zpracovávání symbolu a. Pokud symbol a ještě nebyl zpracováván, je ponechán na své pozici.

Funkce metody bude předvedena na následujícím příkladu: Mějme abecedu $A = \{a, b, c, d, e, f\}$ a posloupnost vstupních symbolů *ebbcaae*. Předpokládejme, že algoritmus zpracovává druhý výskyt symbolu e ve vstupní posloupnosti. Protože symboly c a d byly zpracovávány od posledního zpracování symbolu e nejvýše jednou, bude symbol e v abecedě A přesunut před symbol c.

2.10 Kódování délek sledů

Metoda kódování délek sledů (run rength encoding – RLE) [11] je jednoduchá kompresní metoda využívaná při kompresi textu a obrazu. Její princip je takový, že nahrazuje sledy shodných symbolů jako dvojici: $\langle$*počet výskyt symbolu*$\rangle\langle$*symbol*$\rangle$. Při dekompresi je přečten počet výskytů a výstupem bude tento počet symbolů.

Otázkou, která nastává, je, od jakého počtu shodných symbolů komprimovat. Při komprimování řetězce *abcd* bychom dostali řetězec $1a1b1c1d$, který je delší než původní kvůli přidání počtu výskytů. Proto se určuje, od kolika shodných symbolů se začne komprimovat. Toto číslo je prahem komprimace. Pro text se většinou volí hodnoty 3 nebo 4.

Další otázkou při využití této metody je, jak odlišit počet výskytů symbolu od vstupních symbolů. Například řetězec $2ddddhikkk----00fffffff....$ nelze nahradit řetězcem $24dhi3k5-206f4.$, protože číslo 2 je zde také jedním ze symbolů a při dekompresi by nebyl výstupem původní řetězec. Jedním řešením může být použití speciálního symbolu, například @, který je vložen před počet výskytů, a signalizuje tím, že bude následovat počet výskytů

symbolu, a ne symbol vstupních dat. Poté by náš řetězec $2dddd hikkk----00fffffff....$ byl zakódován jako $2@4dhi@3k@5--@20@6f@4..$ Tento řetězec už lze korektně dekódovat. Problémem tohoto řešení je, že tento speciální symbol se nesmí vyskytovat ve vstupním toku, což vždy nemusí být možné. Dalším řešením může být uložení určitého počtu stejných symbolů a po něm další počet výskytů. Počet symbolů, které se vkládají před počet výskytů je poté prahem určujícím při kolika výskytech stejných symbolů se provede komprese. Pokud bychom měli řetězec $2dddd hikkk----00fffffff....$, při prahu 3 by byl zakódován jako $2ddd1hikkk0---100fff3...1.$

Dalšími problémy může být, že standardní text neobsahuje mnoho opakování, nebo že počet opakování je omezen na velikost jednoho bytu.

Předpokládejme řetězec N znaků ke komprimaci, řetězec obsahuje M opakování s průměrnou délkou L. Každé z M opakování se nahradí třemi symboly (změna, počet a symbol). Kompresní faktor, který poskytuje metoda RLE, je dán pomocí následující rovnice:

$$Kompresní\ faktor\ RLE = \frac{N}{N - M \cdot (L - 3)} \tag{2.11}$$

Existuje více variant tohoto algoritmu. Jednou variantou může být kódování digramů [11], která je vhodná v případě, kdy se ve vstupním toku vyskytují pouze určité znaky, např. pouze číslice, písmena a interpunkce. Poté lze běžně se vyskytující dvojice (digramy) nahradit jedním znakem, který se ve vstupním toku nevyskytuje, např. řídícím znakem kódu ASCII. Další variantou je substituce vzorků [11]. Tato varianta je vhodná pro kompresi počítačových programů, ve kterých se často vyskytují klíčová slova programovacího jazyka, např. while, print apod. Tato slova jsou nahrazována pomocí řídících znaků. Další variantou je relativní kódování [11]. Používá se v případech, kdy se ve vstupním toku nachází řetězce čísel, které se od sebe příliš neliší, nebo vzájemně podobně řetězce. Pro taková data se na výstup odešle pouze první hodnota, a poté se posílají rozdíly. Protože rozdíly jsou pro taková vstupní data malé, stačí k jejich zakódování nižší počet bitů. Pokud se však vyskytne velký rozdíl, je místo něj poslána skutečná hodnota, čímž nastane problém, jak odlišit rozdíly a skutečné hodnoty. Řeší se to tím, že kompresor ke každému odeslanému číslu přidává zvláštní bit (příznak), který rozlišuje, zda se jedná o hodnotu nebo o rozdíl. Kompresor tyto bity akumuluje a čas od času je pošle do dekompresoru. Vysílá-li se každý rozdíl jako byte, musí dekompresor hlídat skupiny osmi bytů a přiřazovat jim příznaky z bytu příznaků.

2.11 Entropické kódování

Pro kompresi výstupu global structure transformation a kódování délek sledů mohou být použity různé entropické kodéry. Nejdůležitějšími jsou Huffmanovo kódování a aritmetické kódování. Výhodou Huffmanova kódování je vyšší rychlost, aritmetické kódování zase nabízí lepší kompresní poměr [2].

Další možností je využít kontextových metod komprese. Tyto metody při přidělování pravděpodobností symbolů závisející nejen na četnosti jejich výskytu, ale také jejich kontextu. Kontextem symbolu je myšleno N symbolů, které mu předcházejí, a využívají se k predikci. Nejznámějšími zástupci kontextových metod jsou metody ze skupiny PPM (Prediction by partial matching).

Kapitola 3

Implementace a testování

V této kapitole bude popsána implementace programu a způsob a výsledky jeho testování.

3.1 Implementace

Program sestává ze dvou částí: knihovny libsc a konzolové aplikace ukazující použití této knihovny. Program je napsán v jazyce C++ s využitím procedurálního a modulárního postupu programování. Byla také vytvořena s ohledem na její přenositelnost mezi různými systémy.

3.2 Testování

Pro měření kompresní výkonnosti se využívá veličin popsaných v kapitole 2.2. Aby se daly kompresní metody objektivně srovnávat, musí se testovat na stejných vstupních datech, jejichž charakter by měl být složen z více typů dat, aby nebyly určité metody zvýhodněny. Proto vznikly tzv. korpusy. Korpus se skládá z více souborů, které byly vybrány tak, aby obsáhly více typů dat (např. text, obraz, apod.), a aby jejich charakter odpovídal standardně používaným souborům. Příklady korpusů jsou Calgary, Canterbury a Silesia.

3.2.1 Calgary korpus

Calgary korpus [7] byl vytvořen v roce 1987 na univerzitě Calgary. Sestává ze 14 souborů, které byly vybrány intuitivně. Dnes je už relativně zastaralý, přesto je stále užitečný k měření výkonnosti kompresních algoritmů a k porovnávání výsledků se staršími algoritmy. Obsah korpusu ukazuje tabulka 3.1.

3.2.2 Canterbury korpus

Canterbury korpus [5] byl vytvořen v roce 1997 na univerzitě Canterbury jako alternativa ke korpusu Calgary, který již byl v té době zastaralý. Dalším důvodem bylo, že se při vývoji kompresních algoritmů využíval korpus Calgary k měření výkonnosti, a tak tyto algoritmy mohly být i neúmyslně vyladěny pro dokumenty v tomto korpusu. Dokumenty v korpusu Canterbury byly na rozdíl od Calgary vybírány z mnoha kandidátů, aby reprezentovaly „průměrné“ dokumenty dané třídy. Obsah korpusu ukazuje tabulka 3.2.

Soubor	Popis	Velikost [B]
bib	Bibliografie (formát referencí)	111 261
book1	Fiktivní kniha	768 771
book2	Ne-fiktivní kniha (formát troff)	610 856
geo	Geofyzikální data	102 400
news	USENET dávkový soubor	377 109
obj1	Objektový kód pro VAX	21 504
obj2	Objektový kód pro Apple Mac	246 814
paper1	Technický dokument	53 161
paper2	Technický dokument	82 199
pic	Černobílý faxový obrázek	513 216
progc	Zdrojový kód v jazyce C	39 611
progl	Zdrojový kód v jazyce LISP	71 646
progp	Zdrojový kód v jazyce PASCAL	49 379
trans	Zápis terminálového sezení	93 695
Celková velikost korpusu		3 141 622

Tabulka 3.1: Soubory obsažené v korpusu Calgary, jakého jsou typu a jejich velikosti.

Soubor	Popis	Velikost [B]
alice.txt	Anglický text	152 089
asyoulik.txt	Shakespeareova hra	125 179
cp.html	Zdrojový kód v jazyce HTML	24 603
fields.c	Zdrojový kód v jazyce C	11 150
grammar.lsp	Zdrojový kód v jazyce LISP	3 721
kennedy.xls	Tabulky v Excelu	1 029 744
lcet10.txt	Technický dokument	426 754
plrabn12.txt	Poezie	481 861
ptt5	Faxové obrázky	513 216
sum	Spustitelné soubory SPARC	38 240
xargs.1	Manuálové stránky GNU	4 227
Celková velikost korpusu		2 816 188

Tabulka 3.2: Soubory obsažené v korpusu Canterbury, jakého jsou typu a jejich velikosti.

3.2.3 Silesia korpus

Korpus Silesia [9] byl vytvořen v roce 2003 na univerzitě Silesia. Záměrem při vytvoření bylo poskytnutí typických souborů používaných v dnešní době. Starší korpusy neobsahovaly soubory větší velikosti, v dnešní době se ovšem pracuje se soubory podstatně větších velikostí než mají soubory v nich obsažené. Proto jsou soubory v korpusu Silesia podstatně větší (6–51MB). V dnešní době jsou také nejpoužívanějšími a nejrychleji se rozšiřujícími typy souborů multimediální soubory a databáze. Z tohoto důvodu jsou zde obsaženy také tři databázové soubory. Multimediální data jsou obyčejně komprimována pomocí ztrátových metod, proto zde nejsou obsažena. Jedinými zástupci multimediálních dat jsou medicínské obrazy, u nichž nelze ztrátové metody použít. Obsah korpusu ukazuje tabulka 3.3.

Soubor	Popis	Velikost [B]
dickens	Shromážděné práce Charlese Dickense	10 192 446
mozilla	Spustitelné soubory Mozzila 1.0 zabalené pomocí tar	51 220 480
mr	Obrázek magnetické rezonance	9 970 564
nci	Databáze chemických struktur	33 553 445
ooffice	DLL knihovna z OpenOffice.org 1.01	6 152 192
osdb	Vzorek MySQL databáze	10 085 684
reymont	Text knihy v polském jazyce ve formátu pdf	6 627 202
samba	Zdrojové kódy Samba2-2.2 zabalené pomocí tar	21 606 400
sao	Katalog binárních záznamů o hvězdách	7 251 944
webster	Slovník anglického jazyka uložený v HTML	41 458 703
xml	Shromážděné XML soubory	5 345 280
x-ray	Obrázek rentgenu	8 474 240
Celková velikost korpusu		211 938 580

Tabulka 3.3: Soubory obsažené v korpusu Silesia, jakého jsou typu a jejich velikosti.

3.3 Testovací postup

Postup komprese u knihovny libsc využívá řetězec metod BWT ⇒ GST ⇒ RLE ⇒ EC. Je načten blok vstupních dat, na kterém je provedena Burrowsova-Wheelerova transformace. Poté jsou data přetransformována pomocí určené metody global structure transformation, dále je provedena metoda kódování délek sledů a nakonec jsou data zakomprimována pomocí aritmetického kódování.

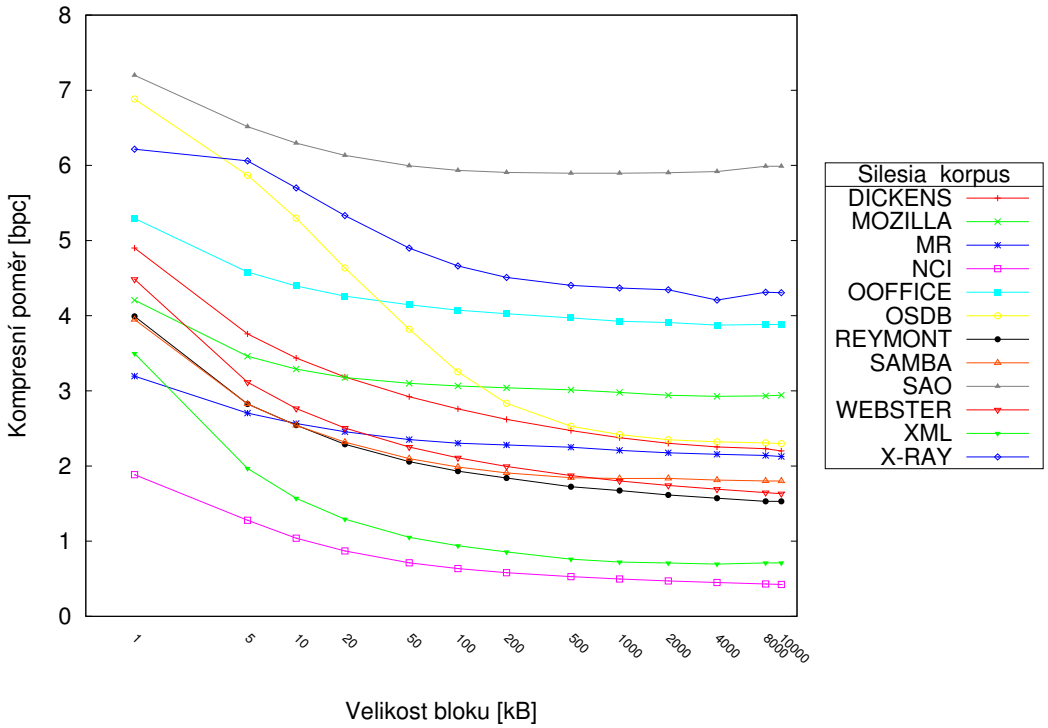
3.4 Výsledky testování

Testování probíhalo zejména na korpusu Silesia, kvůli zastaralosti korpusů Calgary a Canterbury, a také kvůli daleko větší velikosti souborů v korpusu, které více odpovídají v dnešní době běžně používaným souborům. Testování probíhalo na počítači s procesorem Core2Duo 2,26 GHz, pamětí o velikosti 4 GB a operačním systémem Windows 7 32-bit. Pokud není uvedeno jinak, je pro testy zvolena velikost načítaného bloku 500 kB, která je vhodným kompromisem mezi kompresním poměrem, rychlostí a paměťovou náročností.

3.4.1 Vliv velikosti bloku dat při Burrowsově-Wheelerově transformaci

Prvním testem je vliv velikosti načítaného bloku dat. Testování probíhalo na souborech korpusu Silesia, ve fázi global structure transformation byla použita nemodifikovaná metoda přesuň na začátek. Metoda Burrowsovy-Wheelerovy transformace vytváří ve zpracovávaném bloku dat shluky stejných symbolů, tedy čím větší blok, tím větší by měla být pravděpodobnost vytvoření větších shluků symbolů. Podle výsledků v tabulce B.1 a grafu 3.1 lze vyčíst, že tomu tak ve skutečnosti doopravdy je, protože čím větší je blok načítán, tím lepší je kompresní poměr. Lze si však všimnout také toho, že s navyšováním velikosti bloků při velkých velikostech bloků už není zlepšování kompresního poměru tak markantní. Další věcí je, že pokud už velikost bloku přesáhne velikost souboru, kompresní poměr už se nemění. Tento fakt je zřejmý mezi hodnotami pro velikost bloku 8 000 kB a 10 000 kB u souborů ooffice, reymont, sao a xml (jejich velikost je naprosto stejná). Zajímavé ovšem

je, že někdy dochází při zvyšování bloku ke zhoršení kompresního poměru, např. u souboru sao se kompresní poměr při zvětšování velikosi bloku nad 1 000 kB pouze zhoršuje.



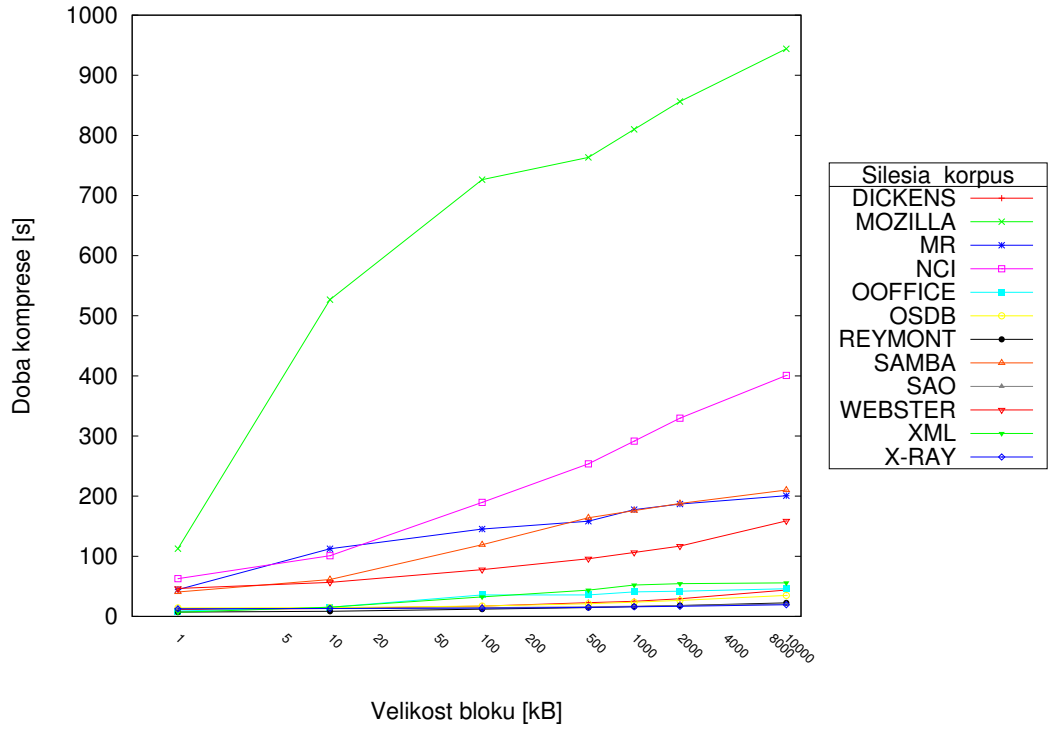
Obrázek 3.1: Kompresní poměry v [bpc] pro různé velikosti bloku a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

Druhým testem je test závislosti doby komprese na velikosti načítaného bloku. Testování probíhalo na souborech korpusu Silesia, ve fázi global structure transformation byla použita nemodifikovaná metoda přesuň na začátek. Protože ve fázi Burrowsovy-Wheelerovy tranformace je třeba vždy seřadit data, a algoritmy pro řazení dat mají horší složitost než lineární, lze očekávat, že při větších blocích bude komprese stejného souboru trvat delší dobu. Výsledky v tabulce B.2 a grafu 3.2 toto tvrzení dokazují. Se zvyšující se velikostí načítaného bloku doba potřebná ke kompresi narůstá. Nejvíce patrné je to u souborů mozilla a nci, kdy je nárůst opravdu rapidní.

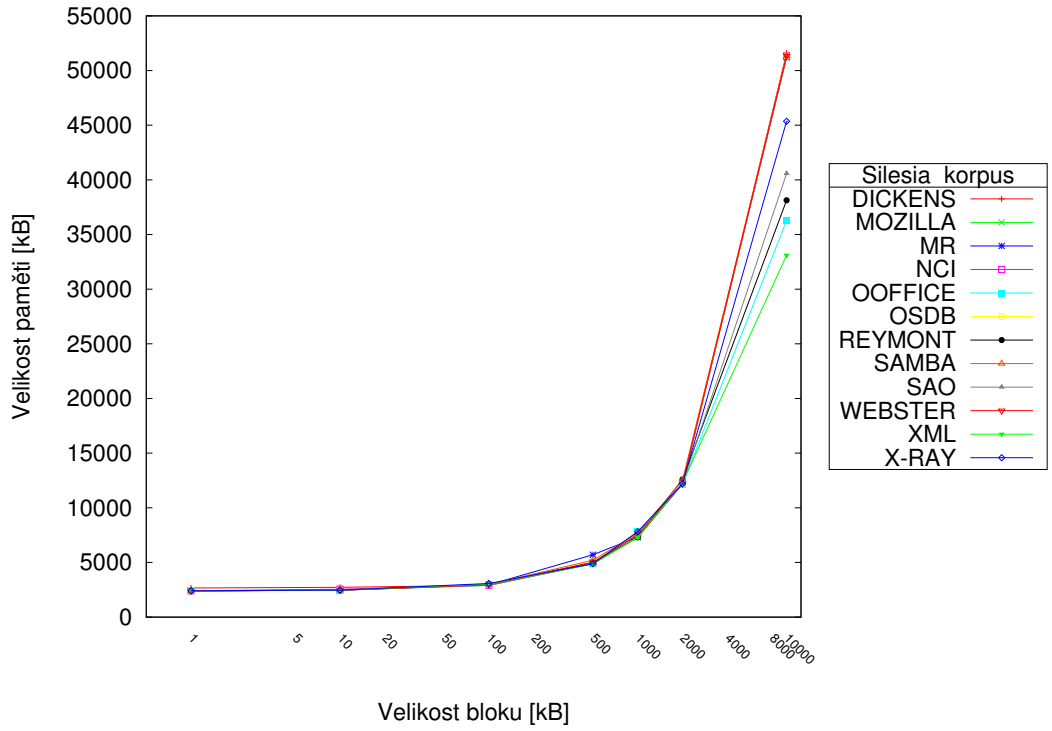
Posledním testem týkajícím se různé velikosti načítaného bloku je test paměťové náročnosti. Testování probíhalo na souborech korpusu Silesia, ve fázi global structure transformation byla použita nemodifikovaná metoda přesuň na začátek. Výsledky testu jsou vyneseny v grafu 3.3. S rostoucí velikostí načítaného bloku také roste paměťová náročnost aplikace. Opět zde podobně jako u testu kompresních poměru platí, že pokud se velikostí bloku přesáhne velikost souboru, paměťová náročnost dále nenarůstá.

3.4.2 Vliv metod přesun na začátek

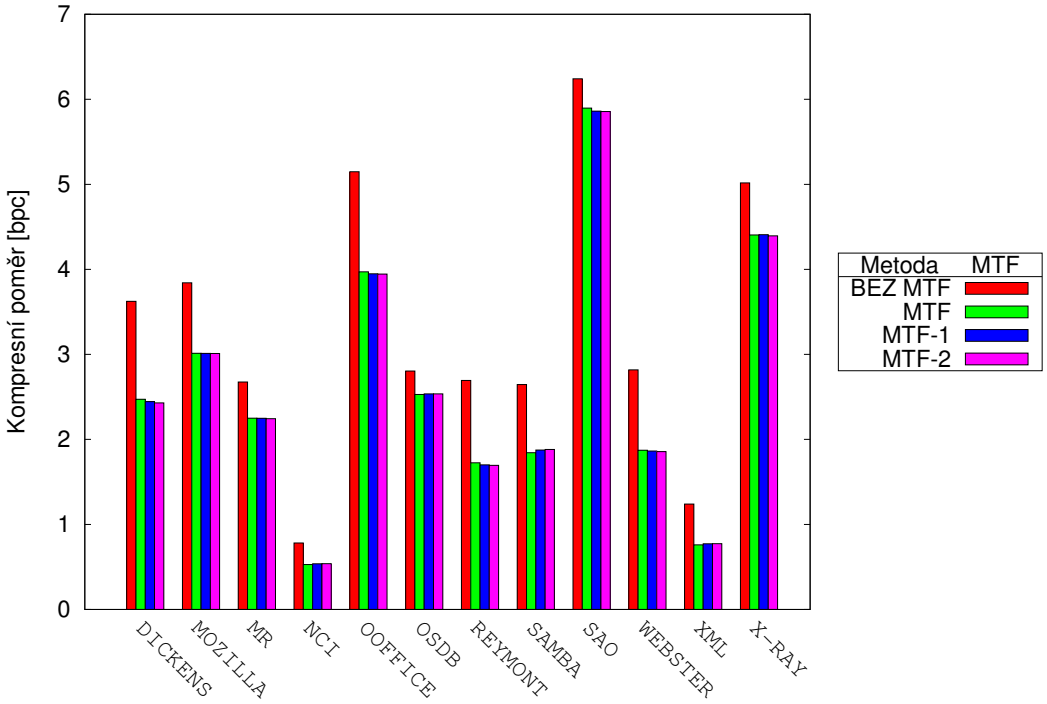
Vliv metody přesun na začátek byl testován na korpusu Silesia při velikosti bloku 500 kB. Byly otestovány i dvě modifikace této metody (MTF-1 a MTF-2) a také jaký vliv má úplné vypuštění této metody z kompresního řetězce. V tabulce B.3 jsou uvedeny výsledky testu. Z výsledků je patrné, že použití metody přesun na začátek značně vylepší kompresní poměr, ale rozdíly mezi jejími modifikacemi jsou minimální. Výsledky jsou znázorněny v grafu 3.4.



Obrázek 3.2: Doba komprese v [s] pro různé velikosti bloku a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím rychlejší je komprese.



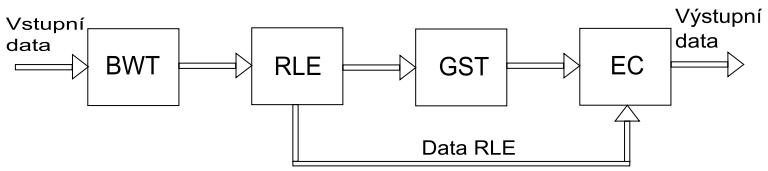
Obrázek 3.3: Spotřeba paměti v [kB] pro různé velikosti bloku a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím menší je paměťová náročnost.



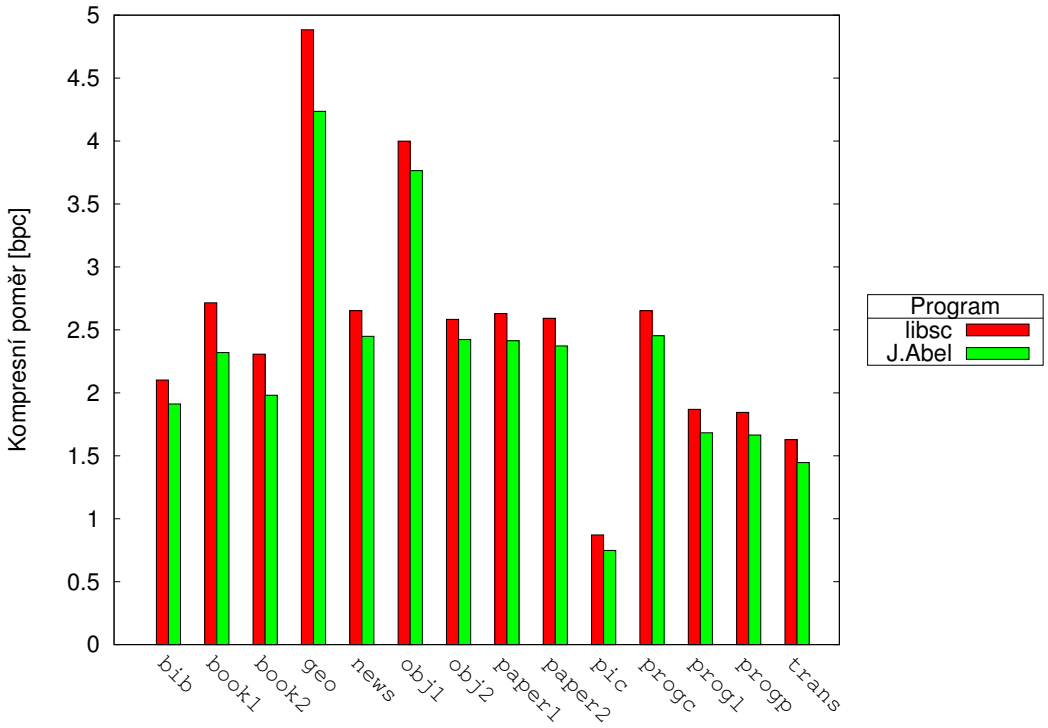
Obrázek 3.4: Kompresní poměry v [bpc] při různých použitích metody přesuň na začátek a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

3.4.3 Srovnání knihovny libsc s výsledky J. Abela

Srovnání probíhalo na korpusu Calgary. Z knihovny libsc byla pro fázi global structure transformation vybrána metoda přesun na začátek a zvolena velikost bloku 500 kB. Podle tabulky B.4 a grafu 3.6 program J. Abela vykazuje lepší kompresní poměry. Je tomu tak proto, že knihovna libsc využívá standardního schématu naznačeného v obrázku 2.2, zatímco zatímco J. Abel ve svém programu využil jiný postup, při kterém je metoda kódování délek sledů použita ještě před metodou přesuň na začátek. Výstup metody kódování délek sledů poté sestává ze dvou částí: hlavní částí obsahující vstupní symboly bez délek jejich sledů, aby nebyl narušen jejich kontext, jdoucí na vstup metody přesuň na začátek a částí obsahující délky sledů vstupních symbolů jdoucí přímo do entropického kodéru, kde je zakódována odděleně od výstupu metody přesuň na začátek. Tento postup ukazuje obrázek 3.5.



Obrázek 3.5: Schéma kompresního řetězce založeného na Burrowsově-Wheelerově tranformaci upraveného J. Abelem.



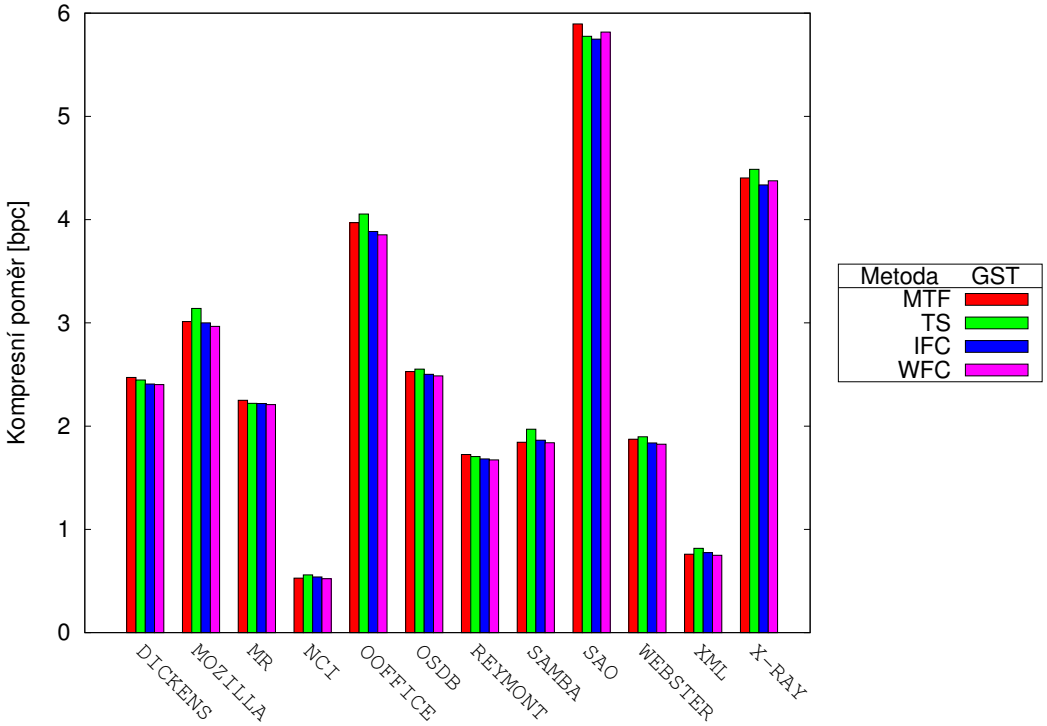
Obrázek 3.6: Srovnání kompresního řetězce knihovny libsc (obrázek 2.2) a postupu J. Abela (obrázek 3.5). Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Calgary. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

3.4.4 Srovnání různých metod fáze global structure transformation

Zde je srovnání různých metod fáze global structure transformation na korpusu Silesia. Velikost bloku je zvolena u všech metod na 500 kB. Pro metodu incremental frequency count byly na základě testování, jehož výsledky jsou v tabulkách B.9, B.10, B.11, B.12 a grafu B.1, nastaveny následující parametry: $windowsize = 32$, $DM = 8$, $threshold = 64$ a $q = 128$. Pro metodu weighted frequency count byla zvolena váhová funkce s parametrem $t_{max} = 2048$. Podle výsledků v tabulce B.5 a grafu 3.7 dosahuje nejlepších kompresních poměrů metoda weighted frequency count a nejhorší je metoda časové razítko. Průměrné dosažené kompresní poměry pro korpus Silesia byly pro metodu přesuň na začátek 2,605, pro metodu časové razítko 2,635, metoda incremental frequency count dosáhla hodnoty 2,565 a metoda weighted frequency count 2,560.

Další částí srovnání metod fáze global structure transformation je jejich časová náročnost. Pro všechny metody byla nastavena velikost načítaného bloku na 500 kB. Podle výsledků v tabulkách B.6 a B.7 a grafech 3.8 a 3.9 je nejrychlejší metoda přesuň na začátek. Metoda incremental frequency count je lehce pomalejší, metoda časového razítka je v současné implementaci podstatně pomalejší a metoda weighted frequency count měla nejhorší časy pro kompresi, při dekompresi však byla rychlejší než metoda časového razítka. Také je vidět, že komprese trvá podstatně delší dobu než dekomprese, je tedy ověřeno, že tento kompresní postup je asymetrický.

V této sekci byly porovnány metody fáze global structure transformation na datovém korpusu Silesia. Z dosažených výsledků se jako nejlépe využitelná jeví metoda incremental frequency count, která dosahuje velmi dobrých kompresních poměrů při zachování nízké



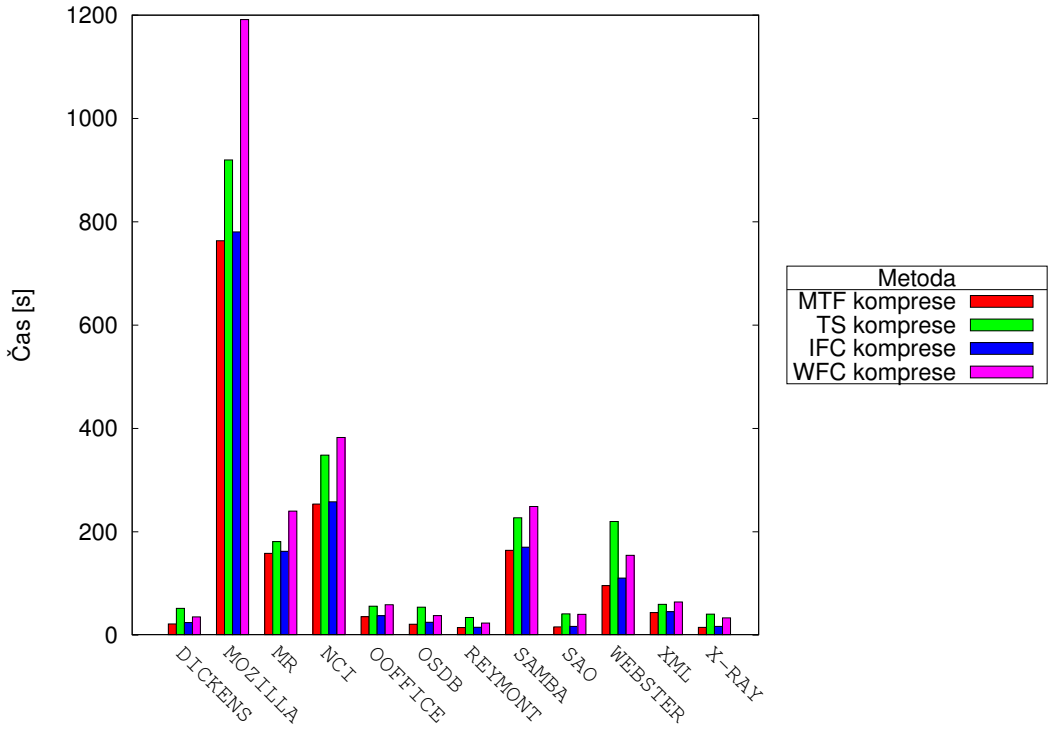
Obrázek 3.7: Kompresní poměry při použití různých metod fáze global structure transformation. Kompresní poměry jsou uvedeny v [bpc] pro celý korpus Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

časové složitosti. Také je u ní možné přizpůsobit se typu dat pomocí změny parametrů používaných při jejím provádění. Metoda weighted frequency count dosahuje nejlepších kompresních poměrů, je však nejpomalejší. Její využití je tedy v případech, kdy záleží hlavně na co nejlepším kompresním poměru a doba komprese nehraje roli. Lze ji také modifikovat pomocí využití různých váhových funkcí. Metoda přesuň na začátek pracovala nejrychleji, lze ji tedy využít v případech, kdy je důležitá rychlost komprese nebo propustnost kompresního řetězce. Její výhodou je také jednoduchá implementace. Přitom dosahuje dobrých kompresních poměrů, u souborů nci, samba a x-ray předčila i metodu incremental frequency count. Metoda časové razítko dosahuje nejhorších kompresních poměrů a v současné implementaci je také velmi pomalá, nelze ji tedy příliš doporučit.

3.4.5 Srovnání knihovny libsc s jinými kompresními programy

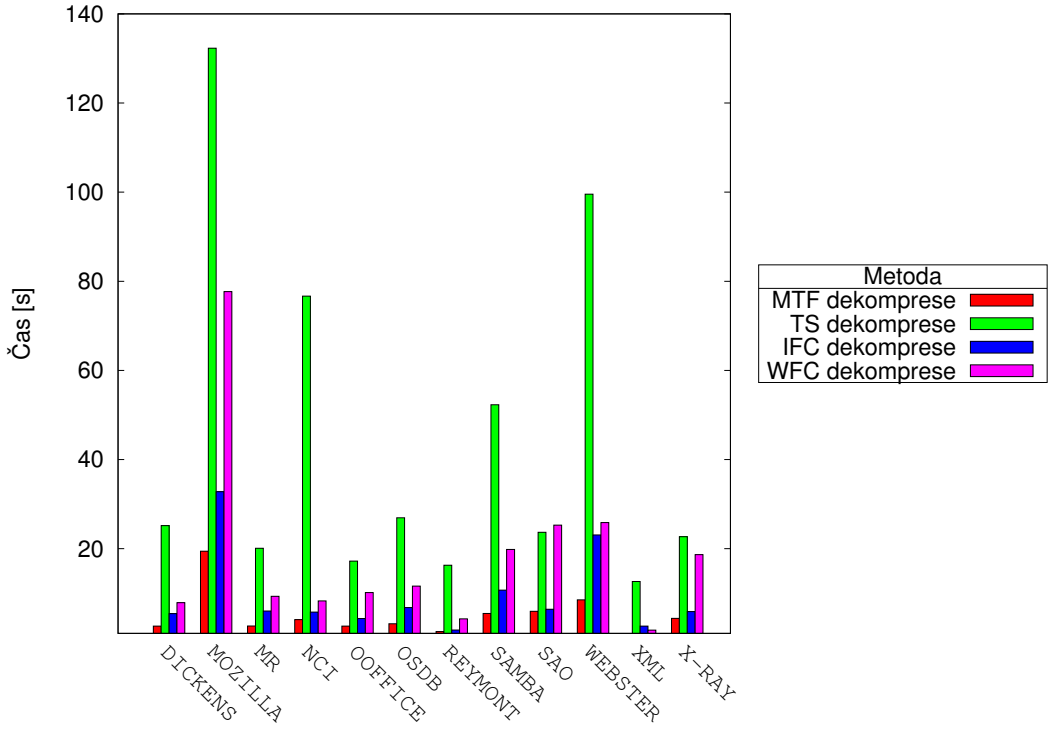
V této sekci je řetězec knihovny libsc dosahující nejlepších výsledků srovnán spolu s jinými kompresními programy. Jako metoda fáze global structure transformation byla zvolena metoda weighted frequency count s parametrem $t_{max} = 2048$. Velikost bloku byla zvolena na 10 000 kB. Z kompresních programů byly vybrány následující:

RAR Program WinRAR 3.90 při nastavení komprese *normální*
BZIP2 Program BZIP2 při nastavení velikosti bloku na 500 kB
ZIP Program zip používá algoritmus DEFLATE, nastavena velikost slovníku na 32 kB
LZMA Program LZMA, nastavena velikost slovníku na 16 MB
PPMd Program PPMd, nastavena velikost slovníku na 16 MB

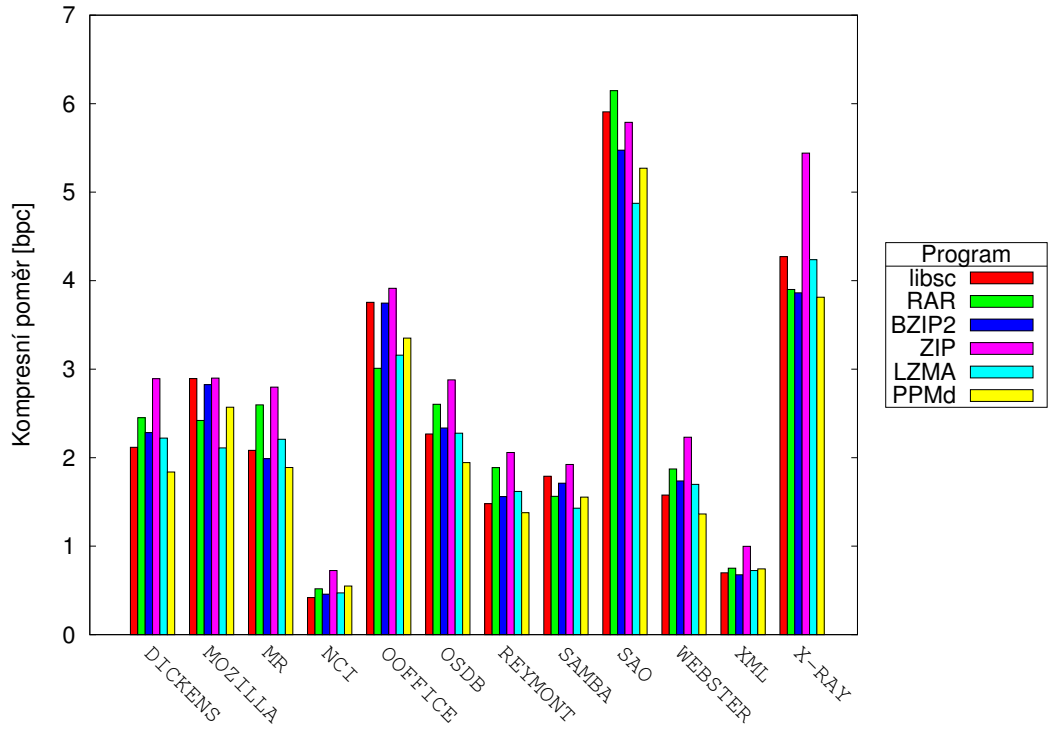


Obrázek 3.8: Srovnání doby trvání komprese různých metod fáze global structure transformation. Časy jsou uvedeny v sekundách. Čím nižší je tato hodnota, tím rychlejší komprese je.

Výsledky testu jsou uvedeny v tabulce B.8 a grafu 3.10. Z výsledků vyplývá, že knihovna libsc je co se kompresního poměru týče konkurenceschopná v porovnání s ostatními běžně používanými kompresními programy, které mnohdy předčila a nejhorší kompresní poměr měla pouze u souboru mozilla. U souboru nci dosáhla dokonce nejlepších výsledků ze všech. Ovšem pokud by se porovnávaly rychlosti komprese, výsledek by byl mnohem horší kvůli pomalému řazení symbolů ve fázi Burrowsovy-Wheelerovy transformace, čímž je knihovna omezena pro běžné používání.



Obrázek 3.9: Srovnání doby trvání dekomprese různých metod fáze global structure transformation. Časy jsou uvedeny v sekundách. Čím nižší je tato hodnota, tím rychlejší dekomprese je.



Obrázek 3.10: Kompresní poměry knihovny libsc v porovnání s jinými kompresními programy. Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

Kapitola 4

Závěr

Cílem této práce bylo seznámit se s metodami komprese dat se zaměřením na statistické metody a vytvoření přenositelné knihovny, ve které budou implementované vybrané metody. Problematiku jsem nastudoval a popsal základní pojmy komprese dat a nejvýznamnější metody. Také jsem z vybraných metod naimplementoval knihovnu, která se jmenuje libsc, a lze ji spolu s demonstračním programem staticcomp stáhnout na adrese https://sourceforge.net/projects/staticcomp/.

Knihovnu jsem testoval na datových korpusech Calgary a Silesia. Kompresní postup implementovaný knihovnou je založen na Burrowsově-Wheelerově transformaci. Pomocí testů jsem dokázal, že při tomto postupu velmi záleží na velikosti načítaného bloku dat. V dnešní době, kdy se kapacita operačních pamětí a výkon počítačů zvyšuje, lze načítat velmi velké bloky a tím dosáhnout dobrých kompresních poměrů.

Také jsem provedl srovnání metod fáze global structure transformation mezi sebou. Z testů vyplynulo, že metoda weighted frequency count dává nejlepší kompresní poměry a metoda přesuň na začátek zase pracuje nejrychleji, což může být v určitých případech důležitým faktorem. Knihovnu jsem nakonec porovnal s běžně používanými kompresními programy, a dokázal jsem její konkurenceschopnost v otázce dosahovaných kompresních poměrů.

Při dalším vývoji knihovny by bylo vhodné vylepšit implementaci fáze Burrowsovy-Wheelerovy transformace, která v současné podobě k řazení využívá algoritmus quicksort a je velmi pomalá, čímž omezuje využitelnost knihovny. Také se nabízí možnost přidat další metody fáze global structure transformation nebo vylepšit stávající kompresní řetězec jiným umístěním metody kódování délek sledů a dále tak vylepšit dosahovaný kompresní poměr.

Literatura

[1] ABEL, J.: A fast and efficient post BWT-stage for the Burrows-Wheeler Compression Algorithm. In *Proceedings of the IEEE Data Compression Conference 2005*, 2005, str. 449.

[2] ABEL, J.: Post BWT stages of the Burrows-Wheeler compression algorithm. *Software: Practice and Experience*, ročník 40, 2010: s. 751–777.

[3] ALBERTS, S.: Improved randomized on-line algorithms for the list update problem. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995, s. 412–419.

[4] ARNAVUT, S., Z.; MAGLIVERAS: Block Sorting and Compression. In *Proceedings of the IEEE Data Compression Conference 1997*, 1997, s. 181–190.

[5] BELL, T.: The Canterbury Corpus. http://corpus.canterbury.ac.nz/descriptions/, 2001.

[6] BINDER, E.: Distance Coder. http://groups.google.com/groups?selm=390B6254.D5113AD2%40T-Online.de, 2000.

[7] CLEARY, J. G.; BELL, T.; WITTEN, I. H.: The Calgary Corpus. http://corpus.canterbury.ac.nz/descriptions/, 2001.

[8] DEOROWICZ, S.: Second step algorithms in the Burrows-Wheeler compression algorithm. *Software: Practice and Experience*, ročník 32, 2002: s. 99–111.

[9] DEOROWICZ, S.: Silesia compression corpus. http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia, 2011.

[10] ELIAS, P.: Interval and Recency Rank Source Coding: Two On-Line Adaptive Variable-Length Schemes. *IEEE Transactions on Information Theory*, ročník 21, 1987: s. 194–203.

[11] SALOMON, D.: *Data Compression: The Complete Reference*. Springer, 2004, ISBN 0-387-40697-2, 899 s.

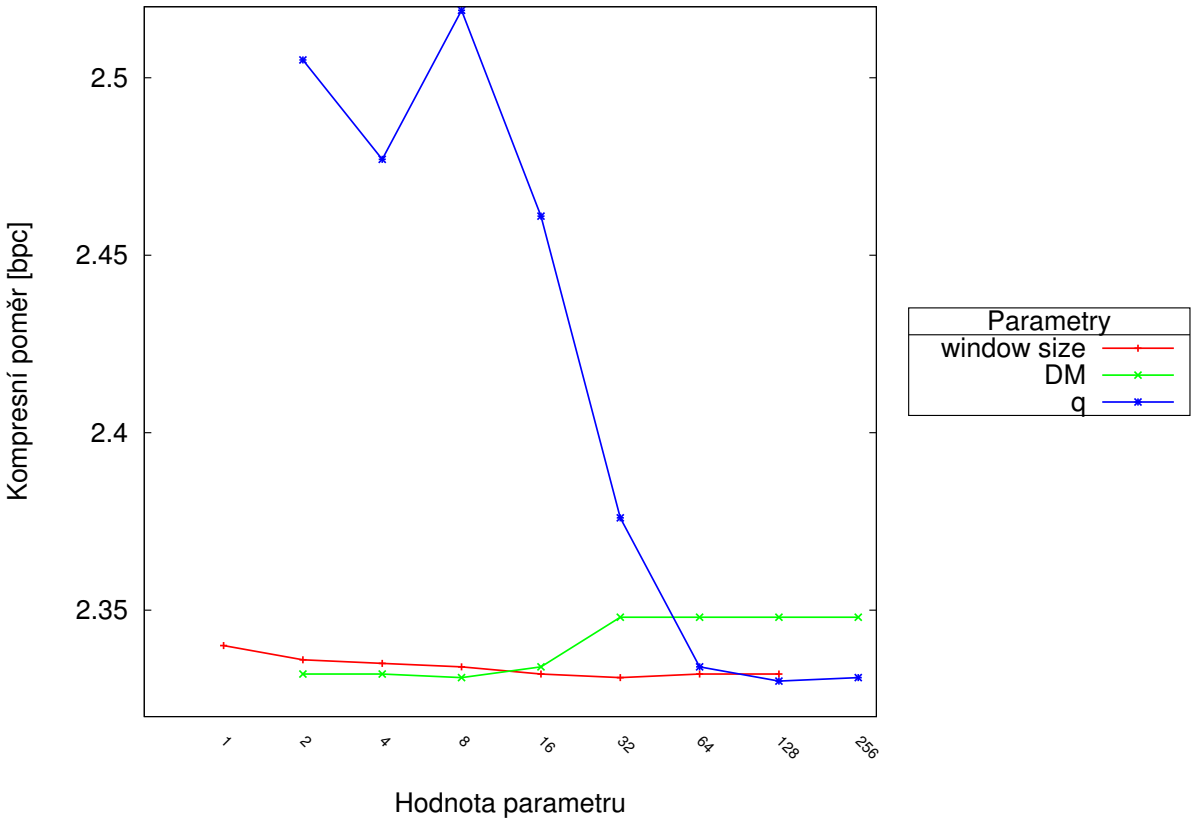
Příloha A

Obsah CD

- **bin** – Adresář se spustitelnými soubory aplikace
- **text** – Adresář se zdrojovými soubory technické zprávy
- **src** – Adresář se zdrojovými soubory aplikace
- **Readme** – Dokumentace k aplikaci
- **Technicka zprava.pdf** – Technická zpráva ve formátu pdf

Příloha B

Výsledky testů



Obrázek B.1: Kompresní poměry při použití různých parametrů metody IFC. Kompresní poměry jsou uvedeny v [bpc] pro celý korpus Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo.

Soubor	Velikost bloku v kB									
	1	5	10	20	50	100	200	500	1 000	2 000
dickens	4,901	3,758	3,437	3,186	2,922	2,760	2,621	2,472	2,377	2,302
mozilla	4,207	3,461	3,290	3,177	3,100	3,065	3,041	3,013	2,979	2,942
mr	3,196	2,704	2,566	2,456	2,352	2,304	2,280	2,250	2,209	2,176
nci	1,885	1,278	1,039	0,870	0,712	0,635	0,581	0,528	0,497	0,471
ooffice	5,295	4,581	4,396	4,260	4,146	4,074	4,027	3,971	3,926	3,909
osdb	6,884	5,869	5,298	4,636	3,820	3,253	2,835	2,529	2,418	2,349
reymont	3,991	2,825	2,543	2,289	2,059	1,932	1,840	1,725	1,673	1,615
samba	3,948	2,826	2,541	2,318	2,099	1,986	1,908	1,844	1,834	1,835
sao	7,199	6,515	6,296	6,133	5,995	5,934	5,906	**5,896**	**5,896**	5,903
webster	4,484	3,115	2,766	2,506	2,253	2,109	1,994	1,873	1,802	1,742
xml	3,497	1,968	1,571	1,293	1,052	0,939	0,857	0,760	0,722	0,709
x-ray	6,216	6,060	5,700	5,332	4,899	4,662	4,509	4,404	4,368	4,345
Průměr	4,642	3,747	3,454	3,205	2,951	2,804	2,700	2,605	2,558	2,525

Soubor	Velikost bloku v kB		
	4 000	8 000	10 000
dickens	2,254	2,231	**2,201**
mozilla	**2,927**	2,934	2,943
mr	2,156	2,140	**2,126**
nci	0,450	0,431	**0,424**
ooffice	**3,876**	3,883	3,883
osdb	2,322	2,308	**2,299**
reymont	1,572	**1,530**	**1,530**
samba	1,813	1,801	**1,800**
sao	5,918	5,988	5,988
webster	1,692	1,646	**1,632**
xml	**0,695**	0,711	0,711
x-ray	**4,208**	4,313	4,306
Průměr	2,490	2,493	**2,487**

Tabulka B.1: Kompresní poměry v [bpc] pro různé velikosti bloku a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně. Jako metoda global structure transformation byla použita nemodifikovaná metoda přesuň na začátek. U souborů ooffice, reymont, sao a xml byla mezi velikostmi bloků 8 000 kB a 10 000 kB překročena jejich velikost, proto kompresní poměr zůstává stejný.

Soubor	Velikost bloku v kB						
	1	10	100	500	1 000	2 000	10 000
dickens	**11,346**	13,142	17,047	22,757	25,088	29,401	4,848
mozilla	**112,691**	526,804	726,449	763,478	810,046	856,356	944,141
mr	**44,505**	112,535	145,282	158,292	177,623	186,895	200,659
nci	**62,841**	100,863	189,633	253,780	291,525	329,699	400,794
ooffice	**8,935**	15,195	35,640	35,681	40,811	41,824	46,005
osdb	**14,259**	14,490	17,234	21,071	23,698	26,544	34,831
reymont	**7,249**	8,533	12,074	14,691	16,290	18,028	22,460
samba	**40,595**	61,382	119,213	164,099	176,009	187,881	210,212
sao	**12,948**	13,060	14,576	15,991	16,890	18,107	20,778
webster	**46,424**	56,651	77,709	95,839	106,350	116,931	158,789
xml	**6,831**	15,279	32,686	43,660	52,144	54,512	55,773
x-ray	**12,563**	13,204	13,673	15,081	16,001	16,845	19,392
Průměr	**31,766**	79,262	116,768	133,702	146,040	156,919	179,807

Tabulka B.2: Doba komprese v [s] pro různé velikosti bloku a pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím rychlejší je komprese. Nejlepší výsledky jsou uvedeny tučně. Jako metoda global structure transformation byla použita nemodifikovaná metoda přesuň na začátek.

Soubor	kompresní poměr [bpc]			
	Bez MTF	MTF	MTF-1	MTF-2
dickens	3,624	2,472	2,444	**2,429**
mozilla	3,842	3,013	3,012	**3,011**
mr	2,675	2,250	2,248	**2,244**
nci	0,782	**0,528**	0,537	0,539
ooffice	5,148	3,971	3,948	**3,945**
osdb	2,804	**2,529**	2,535	2,535
reymont	2,694	1,725	1,701	**1,695**
samba	2,645	**1,844**	1,875	1,882
sao	6,241	5,896	5,861	**5,856**
webster	2,818	1,873	1,863	**1,856**
xml	1,239	**0,760**	0,772	0,774
x-ray	5,017	4,404	4,408	**4,394**
Průměr	3,294	2,605	2,600	**2,597**

Tabulka B.3: Vliv metody přesuň na začátek na kompresní poměr. Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně.

Soubor	kompresní poměr [bpc]	
	libsc	J.Abel
bib	2,102	**1,912**
book1	2,715	**2,320**
book2	2,307	**1,981**
geo	4,884	**4,236**
news	2,652	**2,449**
obj1	3,999	**3,765**
obj2	2,584	**2,423**
paper1	2,630	**2,414**
paper2	2,592	**2,373**
pic	0,871	**0,748**
progc	2,652	**2,454**
progl	1,869	**1,683**
progp	1,845	**1,665**
trans	1,628	**1,446**
Průměr	2,524	**2,276**

Tabulka B.4: Srovnání kompresního řetězce knihovny libsc a postupu J. Abela. Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Calgary. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně.

Soubor	kompresní poměr [bpc]			
	MTF	TS	IFC	WFC
dickens	2,472	2,447	2,408	**2,402**
mozilla	3,013	3,140	3,000	**2,966**
mr	2,250	2,221	2,219	**2,209**
nci	0,528	0,559	0,539	**0,522**
ooffice	3,971	4,054	3,885	**3,853**
osdb	2,529	2,552	2,502	**2,487**
reymont	1,725	1,705	1,683	**1,673**
samba	1,844	1,970	1,864	**1,839**
sao	5,896	5,776	**5,748**	5,817
webster	1,873	1,897	1,837	**1,825**
xml	0,760	0,817	0,755	**0,749**
x-ray	4,404	4,487	4,436	**4,376**
Průměr	2,605	2,635	2,565	**2,560**

Tabulka B.5: Srovnání různých metod fáze global strucutre transformation. Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Silesia. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně.

Soubor	MTF	TS	IFC	WFC
	komprese	komprese	komprese	komprese
dickens	**22,757**	51,846	24,324	35,288
mozilla	**736,478**	919,716	780,396	1191,401
mr	**158,292**	181,063	162,295	240,066
nci	**253,780**	348,376	258,070	382,489
ooffice	**35,681**	55,978	37,608	58,923
osdb	**21,071**	54,116	24,915	37,722
reymont	**14,691**	34,218	15,226	23,462
samba	**164,099**	227,224	170,336	248,951
sao	**15,991**	41,315	16,963	40,214
webster	**95,839**	219,950	110,559	154,474
xml	**43,660**	59,704	45,468	64,139
x-ray	**15,081**	40,677	16,291	33,469
Průměr	**133,702**	186,182	138,590	209,217

Tabulka B.6: Srovnání doby trvání komprese různých metod fáze global strucutre transformation. Časy jsou uvedeny v sekundách. Čím nižší je tato hodnota, tím rychlejší je komprese. Nejlepší výsledky jsou uvedeny tučně.

Soubor	MTF	TS	IFC	WFC
	dekomprese	dekomprese	dekomprese	dekomprese
dickens	**2,620**	25,183	5,426	7,876
mozilla	**19,432**	132,293	32,783	77,679
mr	**2,657**	20,087	6,022	9,324
nci	**4,071**	76,667	5,779	8,272
ooffice	**2,632**	17,215	4,328	10,141
osdb	**3,165**	26,936	6,783	11,612
reymont	**1,399**	16,295	1,757	4,255
samba	**5,465**	52,294	10,711	19,835
sao	**5,960**	23,686	6,411	25,282
webster	**8,518**	99,539	23,094	25,865
xml	**0,768**	12,646	2,636	1,730
x-ray	**4,369**	22,686	5,892	18,658
Průměr	**5,088**	43,794	9,302	18,377

Tabulka B.7: Srovnání doby trvání dekomprese různých metod fáze global strucutre transformation. Časy jsou uvedeny v sekundách. Čím nižší je tato hodnota, tím rychlejší je komprese. Nejlepší výsledky jsou uvedeny tučně.

Soubor	kompresní poměr [bpc]					
	libsc	RAR	BZIP2	ZIP	LZMA	PPMd
dickens	2,117	2,452	2,283	2,892	2,221	**1,838**
mozilla	2,893	2,420	2,825	2,899	**2,110**	2,570
mr	2,083	2,597	1,988	2,797	2,208	**1,889**
nci	**0,419**	0,518	0,458	0,725	0,472	0,549
ooffice	3,755	**3,010**	3,746	3,914	3,158	3,351
osdb	2,268	2,603	2,335	2,879	2,277	**1,944**
reymont	1,480	1,888	1,560	2,058	1,618	**1,378**
samba	1,793	1,563	1,712	1,923	**1,429**	1,554
sao	5,908	6,147	5,475	5,789	**4,874**	5,271
webster	1,578	1,872	1,737	2,231	1,698	**1,364**
xml	0,699	0,751	**0,677**	0,998	0,725	0,743
x-ray	4,272	3,901	3,863	5,441	4,237	**3,813**
Průměr	2,438	2,477	2,388	2,879	2,252	**2,189**

Tabulka B.8: Kompresní poměry knihovny libsc v porovnání s jinými kompresními programy. Kompresní poměry jsou uvedeny v [bpc] pro různé typy souborů korpusu Silesia. Jako metoda fáze global structure byla zvolena metoda weighted frekvenci count, velikost načítaného bloku byla 10 000 kB. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně.

Soubor	Velikost hodnoty window size							
	1	2	4	8	16	32	64	128
dickens	**2,403**	2,405	2,406	2,407	2,407	2,407	2,407	2,407
mr	2,247	2,230	2,222	2,224	2,219	**2,218**	2,219	2,219
nci	0,536	0,536	0,538	**0,533**	0,534	0,536	0,536	0,536
sao	5,763	5,759	5,755	5,755	5,746	**5,743**	5,744	5,744
xml	0,752	0,752	0,752	0,752	0,752	0,752	0,752	0,752
Průměr	2,340	2,336	2,335	2,334	2,332	**2,331**	2,332	2,332

Tabulka B.9: Kompresní poměry v [bpc] pro různé velikosti hodnoty window size pro metodu IFC. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně. Ostatní parametry byly nastaveny následovně: $DM = 16$, $threshold = 256$ a $q = 64$.

Soubor	Velikost hodnoty DM							
	2	4	8	16	32	64	128	256
dickens	2,407	2,407	2,407	2,407	2,407	2,407	2,407	2,407
mr	2,220	2,221	**2,217**	2,224	2,242	2,242	2,242	2,242
nci	0,536	0,535	0,536	**0,533**	**0,533**	**0,533**	**0,533**	**0,533**
sao	**5,745**	5,747	**5,745**	5,755	5,803	5,803	5,803	5,803
xml	0,752	0,752	0,752	0,752	0,752	0,752	0,752	0,752
Průměr	2,332	2,332	**2,331**	2,334	2,348	2,348	2,348	2,348

Tabulka B.10: Kompresní poměry v [bpc] pro různé velikosti hodnoty DM pro metodu IFC. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně. Ostatní parametry byly nastaveny následovně: $windowsize = 8$, $threshold = 256$ a $q = 64$.

Soubor	Velikost hodnoty q							
	2	4	8	16	32	64	128	256
dickens	2,732	2,640	2,747	2,561	**2,407**	**2,407**	**2,407**	**2,407**
mr	2,423	2,446	2,475	2,440	2,376	2,224	**2,215**	2,217
nci	0,579	0,567	0,556	0,558	0,534	**0,533**	0,537	0,535
sao	5,943	5,944	5,964	5,925	5,811	5,755	**5,741**	5,742
xml	0,849	0,788	0,853	8,824	**0,752**	**0,752**	**0,752**	**0,752**
Průměr	2,505	2,477	2,519	2,461	2,376	2,334	**2,330**	2,331

Tabulka B.11: Kompresní poměry v [bpc] pro různé velikosti hodnoty q pro metodu IFC. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně. Ostatní parametry byly nastaveny následovně: $windowsize = 8$, $DM = 16$ a $threshold = 256$.

Soubor	Velikost hodnoty threshold							
	64	128	256	512	1024	2048	4096	8192
dickens	2,407	2,407	2,407	2,407	2,407	2,407	2,407	2,407
mr	**2,217**	2,220	2,224	2,229	2,231	2,230	2,226	2,226
nci	0,539	0,548	0,533	**0,526**	**0,526**	**0,526**	**0,526**	**0,526**
sao	**5,747**	5,750	5,755	5,762	5,767	5,770	5,773	5,775
xml	0,755	0,753	**0,752**	**0,752**	**0,752**	**0,752**	**0,752**	**0,752**
Průměr	**2,333**	2,336	2,334	2,335	2,337	2,337	2,337	2,337

Tabulka B.12: Kompresní poměry v [bpc] pro různé velikosti hodnoty threshold pro metodu IFC. Čím nižší je tato hodnota, tím lepší komprese je dosaženo. Nejlepší výsledky jsou uvedeny tučně. Ostatní parametry byly nastaveny následovně: $windowsize = 8$, $DM = 16$ a $q = 64$.